VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Komponenta výukového serveru TI - Bezkontextové gramatiky

Component of Teaching Server for Theoretical CS - Context-free Grammars

2018 Bc. Jan Havlas

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jan Havlas**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Komponenta výukového serveru TI - Bezkontextové gramatiky
Component of Teaching Server for Theoretical CS - Context-free Grammars

Jazyk vypracování: čeština

Zásady pro vypracování:

V rámci diplomových a bakalařských prací vzniká výukový server pro předměty teoretické informatiky. Jedná se o sadu dynamických webových stránek umožňujících studentům pochopení různých typů úloh a problémů tím, že si mohou zadat na stránce libovolné zadání a zobrazí se jim řešení včetně postupu. Cílem této diplomové práce je vytvořit komponentu pro výuku bezkontextových gramatik.

1. Naprogramujte minimálně následující:
a) zobrazení derivace zadaného slova v zadané gramatice,
b) uživatelem řízené provádění derivace (postupně vybírá, které pravidlo se použije),
c) zobrazení derivačního stromu,
d) redukci gramatiky,
e) převod gramatiky do Chomského a Greibachové normální formy,
f) algoritmus CYK pro zjištění, zda gramatika v CHNF generuje zadané slovo.
2. Cílem není nejefektivnější implementace algoritmů, ale taková, která umožní zobrazovat přehledně jednotlivé kroky včetně vysvětlujících komentářů tak, aby uživatel mohl pochopit princip jejich fungování.
3. Pro jednotlivé algoritmy bude k dispozici alespoň 5 ukázkových vstupů, aby si uživatel mohl postup výpočtu zobrazit i bez zadávání vlastních vstupů.
4. Součástí stránek bude také teoretický popis jednotlivých operací na gramatikách.
5. Zvolte vhodné technologie. Požadavkem je, aby vše běželo jako dynamické webové stránky.

Seznam doporučené odborné literatury:

[1] Dexter C. Kozen: Automata and Computability, Springer, 1997
[1] P. Jančar: Teoretická informatika, VŠB-TU Ostrava 2007 (2010)

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Martin Kot, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.	prof. Ing. Pavel Brandštetter, CSc.
vedoucí katedry	*děkan fakulty*

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 .....

Abstrakt

Cílem této diplomové práce je vytvořit komponentu pro výuku bezkontextových gramatik umožňující studentům pochopit snáze základy bezkontextových gramatik. Zvolené téma jsem vyřešil vytvořením dynamických webových stránek v ASP.NET Web Forms.Vytvořené řešení poskytuje algoritmy řešící redukci bezkontextové gramatiky, derivaci, převod gramatik do Chomského nebo Greibachové normální formy a dále Cocke-Younger-Kasami algoritmus k ověření, zdali lze gramatikou v ChNF generovat zadané slovo a zobrazit derivaci zadaného slova v zadané gramatice. Výsledkem této diplomové práce jsou výukové stránky umožňující studentům zobrazit si postupy jednotlivých implementovaných algoritmů na předpřipravených bezkontextových gramatikách, nebo s možností si zadat vlastní bezkontextovou gramatiku a zobrazit postup algoritmů na zadané gramatice.

Klíčová slova: bezkontextová gramatika, formální gramatika, gramatika, redukce gramatiky, derivace, derivační strom, chomského normální forma, greibachové normální forma, cocke-younger-kasami

Abstract

The main goal of this diploma thesis is to create a component that will help students to be able to learn basics of context-free grammars. I have dealt with this by creating dynamic web pages in ASP.NET Web Forms. This solution provides algorithms for reducing context-free grammars, derivation, grammar conversion to Chomsky's or Greibach's normal form, and the Cocke-Young-Kasami algorithm to verify whether the grammar is in ChNF and can generate a given word and display it's derivation in the specified grammar. The result of this diploma thesis are tutorial pages which allows students to visualize the procedures of each implemented algorithms on preconfigured context-free grammars or after entering their own context-free grammar and to show the algorithm's progression to the given grammar.

Key Words: context-free grammar, formal language, grammar, reduction, derivation, derivation tree, chomsky normal form, greibach normal form, cocke-younger-kasami

Obsah

Seznam použitých zkratek a symbolů

BG	–	Bezkontextová gramatika
CFG	–	Context Free Grammar
CHNF	–	Chomského normální forma
CYK	–	Cocke–Younger–Kasami
FTP	–	File Transfer Protokol
GNF	–	Greibachové normální forma
JS	–	JavaScript

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

V dnešní době je přístup k internetu snadný a studenti si tak mohou dohledat potřebné informace k probíraným předmětům na stránkách školy. Studenti by tedy určitě přivítali možnost, kdyby existovaly výukové stránky, kde by si mohli vyzkoušet řešit příklady k probíranému učivu a zobrazit si na nich jejich postupy řešení. Mohli by si tak snadno osvojit základy probírané látky a umožnit její rychlejší pochopení. To by umožnilo pokračovat rychleji v učivu a probírat další látky, které by se jinak z nedostatku času nemusely stihnout.

Cílem práce je vytvoření komponenty pro výuku bezkontextových gramatik. Výsledkem toho jsou výukové stránky nabízející studentům možnost si vyzkoušet jednotlivé implementované algoritmy na bezkontextových gramatikách a zobrazit si jejich řešení včetně postupu. Výukové stránky budou mít, kromě předpřipravených příkladů, možnost dodatečně vkládat další příklady k jednotlivým algoritmům pomocí úprav příslušných textových souborů na serveru. Na druhou stranu je tu pro studenty možnost si vygenerovat bezkontextovou gramatiku bez nutnosti vkládat vlastní. Kromě toho si studenti budou moci procvičit na příkladech odstranění epsilon a jednoduchých pravidel, nebo redukci bezkontextové gramatiky.

V kapitole 2 jsou popsány formální jazyky, formální gramatiky a bezkontextové gramatiky. Formální jazyky v kapitole 2.1 popisují, co si lze pod nimi představit a které operace lze nad nimi provádět. Formální gramatiky v kapitole 2.2 navazují na předchozí kapitolu 2.1 a kromě definice formálních jazyků popisují derivaci slova s ukázkou na příkladu. V další části dané kapitoly se popisuje rozdělení gramatik do skupin (Chomského hierarchie). Kapitola 3 navazuje na předchozí kapitolu formálních gramatik v kapitole 2.2 a popisuje bezkontextové gramatiky, které jsou typ 2 gramatik viz kapitola 2.2.1. Kromě derivace slova uvádí možné operace, které lze nad zadanými bezkontextovými gramatikami provádět a co jsou to normální formy. Kapitola 4 popisuje použité algoritmy na výukových stránkách (webové aplikaci). Každý algoritmus je vždy zakončen ukázkovým příkladem na kterém lze vidět, jak takový algoritmus funguje. Kapitoly 5, 6 a 7 se pak zabývají analýzou, návrhem a implementaci aplikace. Kapitola 7.5 popisuje sestavení webové aplikace a kapitola 7.6 její nasazení na server. Součásti je i možnost konfigurace (úprav) předpřipravených příkladů na serveru a věnuje se tomu kapitola 8. V kapitole 9 je pak popsána možnost změny zobrazovaných textů v postupu řešení algoritmů na výukových stránkách. V poslední kapitole 10 jsou shrnuty dosažené výsledky práce a možné způsoby, jak vylepšit výukové stránky nebo o jaké funkce je rozšířit do budoucna.

2 Teorie formálních jazyků

2.1 Formální jazyky

Co jsou to formální jazyky [1] a čím jsou tvořeny si lze nejlépe popsat na běžně používaných jazycích. Mezi takový běžně používaný patří např. anglický jazyk. U anglického jazyka víme, že je tvořený abecedou a ta je tvořena malými písmeny [a-z], velkými písmeny [A-Z] a dále pak čísly [0-9] nebo speciálními znaky (+,-,*,/, ...). Abecedu tedy tvoří symboly (znaky) a z daných znaků se dají tvořit slova (řetězce). Slova jako taková lze dále skládat za sebou a tím tzv. tvořit řetězce slov. Z pohledu programováni si lze jazyk představit např. jako programovací jazyk C#, který je tvořen klíčovými slovy: if, else, while, for a operátory, speciálními znaky atd. Řetězením klíčových slov ve správném pořadí (syntaxi daného programovacího jazyka) lze dojít ke kódu, který lze úspěšně sestavit a spustit. K syntaxi programovacího jazyka bych jen v krátkosti uvedl, že se jedná o pravidla, jakým způsobem se má zapisovat zdrojový kód, tak aby byl správně funkční. K popsání formálního jazyka je tedy potřeba popsat abecedu, slova a řetězení daných řetězců (slov). Nyní si popíšeme obecněji jednotlivě zmíněné části.

2.1.1 Abeceda

Abeceda je tvořena konečnou množinou symbolů (znaků, čísel, ...). Podmínkou je, aby abeceda obsahovala alespoň jeden znak a tedy množina nesmí být prázdná.

2.1.2 Slovo

Slovo, nebo jinak řečeno řetězec, je tvořen konečnou posloupnosti symbolů abecedy. Řetězec si označíme jako w a délku daného slova lze pak zapsat jako $|w|$. Pokud bychom tedy měli slovo w=0101, tak délka slova w je $|w|$=4. Dalším výrazem je $|w|_a = i$, kde i je počet výskytů znaků a v daném slově. V případě tedy slova w=01010 je $|w|_1$=2 a $|w|_0$=3. Důležité je pak zmínit slovo délky 0 znaků. Tomuto případu se říká prázdné slovo a obvykle se značí symbolem ε nebo λ. Dalšími používanými pojmy jsou:

- Prefixy slova w=abba
  - jsou všechna tato podslova z w zapsaná v množině X=$\{a, ab, abb, abba\}$
- Sufixy slova w=abba
  - jsou všechna tato podslova z w zapsaná v množině Y=$\{a, ba, bba, abba\}$
- Podslova slova w=abba
  - jsou všechna tato slova z w zapsaná v množině Z=$\{a, b, ab, bb, ba, abb, bba, abba\}$

2.1.3 Jazyk

Jazyk je tvořen konečnou množinou slov. Konečná množina slov je tvořena slovy takovými, podle toho, jaký jazyk (programovací jazyk, anglický jazyk) nebo abecedu (Breilovo písmo, morseovka) zastupují.

Nyní víme z čeho se takový formální jazyk skládá a teď se zaměříme na operace, které lze nad takovými jazyky provádět. Mezi tyto operace patří zřetězení, sjednocení, průnik, rozdíl jazyků, doplněk, zrcadlový obraz a kvocient jazyka. V krátkosti si nyní některé z těchto operací popíšeme:

1. Zřetězení jazyků $L_1 = \{a, aa\}$ a $L_2 = \{b, bb\}$, kde $L = L_1 \bullet L_2 = \{uv | u \in L_1 \wedge v \in L_2\}$
   (a) $L_1 \bullet L_2 = \{ab, abb, aab, aabb\}$
   (b) $L_2 \bullet L_1 = \{ba, baa, bba, bbaa\}$
2. Iterace jazyka $L_1 = \{a\}$, kde jazyk $L^* = L^+ \cup \{\varepsilon\}$ a jazyk $L^+ = L^1 \cup L^2 \cup .. \cup L^n$
   - $L_1{}^* = \{\varepsilon, a, aa, aaa, \ldots\}$
3. Průnik jazyků $L_1 = \{a, aa, ab, ba\}$ a $L_2 = \{ab, ba, bb\}$, kde $L = L_1 \cap L_2 = \{u | u \in L_1 \wedge u \in L_2\}$
   - $L_1 \cap L_2 = \{ab, ba\}$
4. Zrcadlový obraz jazyka $L_1 = \{a, ab, bab, abab\}$, kde $L^R = \{u^R | u \in L\}$ tak, že $(x_1 x_2 \ldots x_n)^R = x_n \ldots x_2 x_1$
   - $L_1^R = \{a, ba, bab, baba\}$
5. Kvocient jazyka $L_1 = \{a, bb, abb, abab\}$ podle jazyka $L_2 = \{ab\}$
   (a) Pravý kvocient $L_1/L_2 = \{ab\}$, kde $L_1/L_2 = \{u | \exists v \in L_2 : uv \in L_1\}$
   (b) Levý kvocient $L_2 \backslash L_1 = \{b, ab\}$, kde $L_2 \backslash L_1 = \{v | \exists u \in L_2 : uv \in L_1\}$

K závěru kapitoly je důležité zmínit, že vznikla u formálních jazyků potřeba zjistit, jestli nějaké slovo do jazyka patří nebo ne. Proto se v další kapitole 2.2 budeme zabývat formálními gramatikami. Pomocí formálních gramatik je totiž možné rozhodnout, jestli lze nějaké slovo generovat zadanou gramatikou a tedy jestli patří nebo nepatří do daného jazyka. K rozhodnutí lze např. použít algoritmy (CYK, Earley, atd.) a další k tomu určené. Pokud použijeme nějaký algoritmus k rozhodnutí hovoříme o tzv. rozhodovacím mechanizmu. Mezi rozhodovací mechanizmy např. patří Turingův stroj nebo konečné automaty, ale ty pro nás v této práci nejsou důležité, a proto nebudou dále rozebírány.

2.2 Formální gramatiky

Formální gramatiky, zkráceně FG, tvoří množina pravidel, neterminálů, terminálů a počáteční symbol. Definice formální gramatiky zní následovně:

Definice 1 *Formální gramatika G je definovaná čtveřicí* (N, T, P, S) *[2], kde*

- N *je neprázdná konečná množina neterminálů*
- T *je konečná množina terminálů tak, že* $T \cap N = \emptyset$
- P *je konečná množina přepisovacích pravidel. Každé pravidlo ve tvaru (T∪N)*N(T∪N)* → (T∪N)**
- $S \in N$ *je počáteční symbol gramatiky.*

Formální gramatiky jako takové umožňují generovat řetězce z počátečního neterminálu za použití pravidel, která gramatiku tvoří. Předtím než se tomu začneme věnovat, je potřeba si popsat co je to terminál a neterminál

1. Neterminály jsou symboly, které se přepisují (nahrazují) na nějaký řetězec pravé strany pravidla, kterému odpovídá nějaká levá strana pravidla a proto se jim často říká proměnné, lze je totiž přepsat (symbol se upravuje).
    - Neterminál – je obvykle značen jako znak velké abecedy (A,B,C,. . . ,Z).
2. Terminály jsou symboly, které se nijak nemění (nelze je přepsat) a jsou často označovány jako tzv. konstanty.
    - Terminál – je obvykle značen jako znak malé abecedy (a,b,c,. . . ,z), číslice (0,. . . ,9) nebo speciální symbol (+,-,*,/,. . .).

Teď víme, jak se definují formální gramatiky a co jsou to terminály a neterminály. Můžeme se tedy pustit do tzv. derivace k ověření, zdali lze nějaké slovo FG generovat nebo ne. Derivaci začínáme vždy od počátečního neterminálu a obvykle tento neterminál označujeme jako S. V následujících krocích se pak pokračuje tak, že se v derivaci přepíši v náhodném nebo vybraném pořadí vyskytující se levé strany pravidel za jejich pravé strany pravidel. V případě, že existují pravidla, které mají levou stranu stejnou, ale pravá strana je pokaždé jiná, vybere se jedno z nich tzv. nedeterministickým způsobem. Takové pravidlo pak nahradí levou stranu v derivaci její pravou stranou. Přepis v derivaci probíhá do doby, než se všechny znaky (levé strany pravidel) a tedy neterminály nepřepíší na nějaký terminální symbol. Pro představu si nyní ukážeme postup na následujícím příkladu 1 na další straně.

Příklad 1
Je zadaná FG, kde N={S, B}, T={a, b}, P={S → aBS,S → ab,Ba → aB,Bb → bb} a neterminál S je počátečním symbolem. Lze FG generovat slovo $w = aaabbb$? (Odpověď zjistíme derivací, kterou začneme z počátečního neterminálu S).
S=>aBS=>$aBaBS$=>$aaBBS$=>$aaBBab$=>$aaBaBb$=>$aaBabb$=>$aaaBbb$=>$aaabbb$
Odpověď: Ano, slovo $w = aaabbb$ lze generovat zadanou gramatikou. ■

Našli jsme derivaci, která po konečném počtu kroků došla ke slovu $w = aaabbb$. Jsme tedy schopni zjistit, jestli lze zadanou formální gramatikou generovat nějaké slovo. V případě, že pro nějaké slovo existuje více možností, jak pomocí levé derivace k němu dojít, tak mluvíme o tzv. nejednoznačné (víceznačné) gramatice. To samé platí i v případě, že existuje stejná možnost pro pravou derivaci. V opačném případě se jedná o gramatiku jednoznačnou, kde existuje pouze jeden způsob, jak dojít pomocí derivace k danému slovu v konečném počtu kroků. Jak zjistit, jestli je gramatika víceznačná, je problém nerozhodnutelný. V textu zazněla levá a pravá derivace a proto si nyní popíšeme tyto typy derivací:

1. Obecná derivace
    - přepisují se v derivaci vyskytující se levé strany pravidel v náhodném pořadí (nedeterministickým způsobem) do doby, než se všechny nepřepíší na nějaký terminální symbol (slovo).
2. Levá derivace
    - přepisuje se v derivaci vždy první vyskytující se levá strana pravidla (směr zleva doprava).
3. Pravá derivace
    - přepisuje se v derivaci vždy poslední vyskytující se levá strana pravidla (směr zprava doleva).

Nyní víme, co jsou to formální gramatiky, jak lze rozhodnout jestli lze nějaké slovo gramatikou generovat nebo ne a jaké typy derivací lze přitom použít (levá, pravá). Dále si v menší podkapitole popíšeme typy gramatik pomocí Chomského hierarchie, mezi které patří bezkontextové gramatiky. Bezkontextové gramatiky jsou hlavním tématem této diplomové práce a po probrání kapitoly 3 se bude možné posunout k další části a to k realizaci aplikace (výukových stránek) používající algoritmy uvedené v kapitole 4.

2.2.1 Chomského hierarchie

V roce 1956 Noam Chomsky jako první klasifikoval gramatiky do skupin a tříd a výsledkem byla Chomského hierarchie [3]. Chomského hierarchie je zobrazena na obrázku 1. Slovní popis jednotlivých typů gramatiky je pak uveden na následující straně.

- Typ 0 – Frázové gramatiky (Rekurzivně spočetné)
  - jedná se o formální gramatiky, které nejsou nijak omezeny
- Typ 1 – Kontextové gramatiky (Kontextové jazyky)
  - Tvar pravidel je $\alpha A\beta \rightarrow \alpha\gamma\beta$ kde
    - A je neterminál
    - α, γ, β jsou řetězce tvořeny terminály a neterminály
  - Řetězec γ nesmí být prázdný, ale výjimka je v případě pravidla $S \rightarrow \varepsilon$
- Typ 2 – Bezkontextové gramatiky (Bezkontextové jazyky)
  - Tvar pravidel je $A \rightarrow \beta$, kde
    - A je neterminál
    - β je řetězec složený z terminálů a neterminálů
- Typ 3 – Regulární gramatiky (Regulární jazyky)
  - Pravidla ve tvaru
    - Lineárně levé: $A \rightarrow xB$ nebo $A \rightarrow x$
    - Lineárně pravé: $A \rightarrow Bx$ nebo $A \rightarrow x$

    (A, B jsou neterminály a x je terminál)[1]

Obrázek 1: Chomského hierarchie

[1] Je možné použít jen jeden typ v jedné gramatice.

3 Bezkontextové gramatiky

3.1 Co jsou to bezkontextové gramatiky

Bezkontextové gramatiky jsou typem 2 Chomského hierarchie. Tak jako formální gramatiky se bezkontextové gramatiky skládají z pravidel, neterminálů, terminálů a počátečního symbolu. Rozdíl mezi těmito gramatikami je ve způsobu zápisu pravidel, kde u bezkontextových gramatik se levá strana každého pravidla skládá pouze z jednoho neterminálního symbolu (viz definice 2).

Definice 2 *Bezkontextová gramatika je definována jako uspořádaná čtveřice* $G = (N, T, P, S)$ *[4], kde*

- N *je neprázdná konečná množina neterminálů*
- T *je konečná množina terminálů tak, že* $T \cap N = \emptyset$ *a*
- P *je konečná množina pravidel typu* $A \to \beta$*, kde*
  - A *je neterminál, tedy* $A \in N$
  - β *je řetězec složený z terminálů a neterminálů, tedy* $\beta \in (N \cup T)^*$.
- $S \in N$ *je počáteční symbol gramatiky.*

Pro představu si na příkladu ukážeme zápis bezkontextové gramatiky s použitím znaku | pro oddělení více pravidel obsahujících stejný levý neterminál a ověříme si, jestli zadaná gramatika generuje nějaké slovo.

Příklad 2
Bezkontextová gramatika G (počáteční neterminál S):
$S \to aSa \mid A$
$A \to b \mid bA$
$B \to a \mid aB$

Otázka: Nachází se slovo $w = aabbbaa \in$ L(G) ?
Levá derivace: $S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aaAaa \Rightarrow aabAaa \Rightarrow aabbAaa \Rightarrow aabbbaa$
Odpověď: ANO, slovo $w = aabbbaa \in$ L(G) ■

Při přečtení gramatiky si čtenář určitě všiml, že neterminál B vyskytující se na levé straně nebyl ani jednou použit a že dokonce ani být použit v derivaci nemůže. Pro gramatiku je tedy zbytečné, aby ho gramatika vůbec obsahovala. Proto je dobré v těchto případech na gramatice provést tzv. redukci, která se postará o zpřehlednění gramatiky a odstranění zbytečných a nepotřebných pravidel. Jak redukce gramatiky probíhá je popsáno v kapitole 4.1. V některých případech se hodí tvary pravidel ještě více omezit bez omezení síly gramatiky. Z tohoto důvodu se používají tzv. normální formy viz kapitola 3.2.

3.2 Normální formy

Jedná se o jasně definovaný zápis (tvar) povolených pravidel bezkontextové gramatiky. Pravidla musí splňovat konkrétní podmínky normální formy a nepovolují se jiná pravidla mimo definována. Platí, že každou bezkontextovou gramatiku lze sérií algoritmických operací převést na příslušnou normální formu zde uvedenou. Výstupem takového převodu je opět gramatika ekvivalentní, tzn. že generuje stále stejný jazyk. To znamená, že gramatika dokáže generovat ta samá slova jako gramatika původní a žádná jiná.

3.2.1 Chomského normální forma

V Chomského normální formě je každá BG, která má všechna pravidla ve tvaru $A \to a$ nebo $A \to BC$, kde a je terminální symbol a A, B, C jsou neterminální symboly. V speciálním případě je povolené i pravidlo $S \to \varepsilon$, ale tento neterminál S musí být počátečním neterminálem BG a zároveň se nesmí vyskytovat nikde jinde v pravidle na pravé straně gramatiky. U gramatiky v ChNF lze zjistit s polynomiální časovou složitostí, zda nějaké slovo lze nebo nelze generovat a to např. pomocí Cocke–Younger–Kasami algoritmu (CYK), který je popsán v kapitole 4.5.

- Ukázka zápisu gramatiky v ChNF:

  $S \to AS|AB$
  $A \to a|AA$
  $B \to b$

3.2.2 Greibachové normální forma

V Greibachové normální formě je každá BG, která má všechna pravidla ve tvaru $A \to a$ nebo $A \to aX_0 \dots X_n$, kde a je terminální symbol a A,$X(X_0,..,X_n)$ jsou neterminální symboly. Jedná se o tzv. pravidla, která začínají terminálním symbolem a pak počtem n neterminálních symbolů, kde počet neterminálů může být i nulový. Výhodou tohoto zápisu pravidel je např. při generování konkrétního slova, kde při každém dalším výběru pravidla použijeme taková, která obsahují následující znak hledaného slova a díky tomu snadněji k němu dojdeme.

- Ukázka zápisu gramatiky v GNF:

  $S \to aS|aSB|a$
  $B \to bB|bBSB|bS$

4 Algoritmy použité na výukových stránkách

4.1 Redukce bezkontextové gramatiky

Redukce bezkontextové gramatiky [5] je postup, při kterém se projdou všechna pravidla gramatiky a odstraní se ta pravidla, kterými nelze generovat alespoň jedno terminální slovo nebo pravidla, která nejsou dostupná z počátečního neterminálu. Pro začátek je potřeba si nadefinovat dvě pomocné množiny. Tyto dvě pomocné množiny se obvykle označují jako množina $\mathcal{T}$ a $\mathcal{D}$. Algoritmus naplňuje nejdříve množinu $\mathcal{T}$ neterminály z pravidel zadané gramatiky a až pak množinu $\mathcal{D}$. To, jak se množiny naplňují, si ukážeme nejdříve za pomocí definice a následně i slovně pro snadnější představu, jak daný algoritmus funguje. Závěrem této kapitoly je ukázkový příklad. Definice pomocných množin $\mathcal{T}$ a $\mathcal{D}$ zní takto:

1. Množina $\mathcal{T}$
   - $\mathcal{T}_1 = \{X | \exists w \in T^* : (X \to w) \in P\}$
   - $\mathcal{T}_{i+1} = \mathcal{T}_i \cup \{X | \exists \alpha \in (\mathcal{T}_i \cup T)^* : (X \to \alpha) \in P\}$
   - $\mathcal{T}_n = \mathcal{T}_{n+1} \Rightarrow \mathcal{T}_n = \mathcal{T}$
2. Množina $\mathcal{D}$
   - $\mathcal{D}_1 = \{S\}$
   - $\mathcal{D}_{i+1} = \mathcal{D}_i \cup \{X | \exists Y \in \mathcal{D}_i, \alpha_1, \alpha_2 \in (N \cup T)^* : (Y \to \alpha_1 X \alpha_2) \in P\}$
   - $\mathcal{D}_n = \mathcal{D}_{n+1} \Rightarrow \mathcal{D}_n = \mathcal{D}$

Slovně by se pak naplňování těchto množin provedlo takto:

1. Vytvoření pomocné množiny $\mathcal{T}$
   - Množina $\mathcal{T}$ bude obsahovat neterminály, které lze pomocí konečného počtu kroků derivace přepsat na terminály.
   - Nejprve do množiny přidáváme neterminály z levých stran pravidel, která mají na pravé straně pouze terminály.
   - Dále přidáváme neterminály z levých stran pravidel, které mají na pravé straně řetězce terminálů a neterminálů z množiny $\mathcal{T}$.
   - Proces opakujeme, dokud do množiny $\mathcal{T}$ máme co přidávat.
   - Procházíme pravidla gramatiky a odstraníme taková pravidla, která obsahují neterminály, jež se nenacházejí v množině $\mathcal{T}$.
2. Vytvoření pomocné množiny $\mathcal{D}$
   - Množina $\mathcal{D}$ bude obsahovat neterminály, které jsou dosažitelné z počátečního neterminálu (mohou se tedy vyskytnout v nějaké derivaci slova generovaného gramatikou).

- Nejprve se do množiny $\mathcal{D}$ přidá počáteční neterminál.
- Dále se přidávají ty neterminály, které se vyskytují na pravé straně pravidel, jejichž levá strana již je v množině $\mathcal{D}$.
- Proces opakujeme, dokud máme co do množiny $\mathcal{D}$ přidávat.
- Procházíme pravidla gramatiky a odstraníme taková pravidla, která obsahují neterminály, jež se nenacházejí v množině $\mathcal{D}$.

Výstupem je redukovaná bezkontextová gramatika. Důležitou podmínkou je, aby redukce začínala vždy naplňováním pomocné množiny $\mathcal{T}$ a následně $\mathcal{D}$ a ne naopak. Změna pořadí by totiž mohla mít v určitých případech za důsledek to, že by výsledná bezkontextová gramatika na výstupu nebyla redukovaná. Závěrem této kapitoly bych chtěl zmínit, že po provedení redukce (naplnění množiny $\mathcal{T}$) u bezkontextové gramatiky jsme schopni říct, jestli zadanou gramatikou lze generovat alespoň jedno slovo. Pokud by totiž množina $\mathcal{T}$ neobsahovala počáteční neterminál (S) víme, že daný neterminál se nedá přepsat na terminální symbol. To by ve výsledku znamenalo to, že v dané gramatice nelze generovat (vytvořit) ani jedno slovo. Samotná redukce by v tomto případě skončila „neúspěšně“, protože první neterminál v množině $\mathcal{D}$ má být počáteční neterminál, ale ten se nenachází v množině $\mathcal{T}$ a proto množina $\mathcal{D}$ zůstane prázdná.

Příklad 3 (Redukce BG)
Bezkontextová gramatika G s počátečním neterminálem S, kde
$S \to aAa$
$A \to Sb|bCC|DaA$
$C \to abb|DD$
$E \to aC$
$D \to aDA$
Postup redukce gramatiky na uvedeném příkladu.

1. Pomocná množina $\mathcal{T}$

   - $\mathcal{T}_1 = \{C\}$
   - $\mathcal{T}_2 = \{C, A, E\}$
   - $\mathcal{T}_3 = \{C, A, E, S\}$
   - $\mathcal{T}_4 = \{C, A, E, S\}$
   - $\mathcal{T}_3 = \mathcal{T}_4 \Rightarrow \mathcal{T} = \mathcal{T}_3$

   Odstranění pravidel s neterminály, které se nenacházejí v množině $\mathcal{T}$:
   $S \to aAa$
   $A \to Sb|bCC|\cancel{DaA}$
   $C \to abb|\cancel{DD}$
   $E \to aC$
   $\cancel{D \to aDA}$

2. Pomocná množina $\mathcal{D}$

   - $\mathcal{D}_1 = \{S\}$
   - $\mathcal{D}_2 = \{S, A\}$
   - $\mathcal{D}_3 = \{S, A, C\}$
   - $\mathcal{D}_4 = \{S, A, C\}$
   - $\mathcal{D}_3 = \mathcal{D}_4 \Rightarrow \mathcal{D} = \mathcal{D}_3$

   Odstranění pravidel s neterminály, které se nenacházejí v množině $\mathcal{D}$:
   $S \to aAa$
   $A \to Sb|bCC$
   $C \to abb$
   ~~$E \to aC$~~

Výsledkem je redukovaná gramatika s pravidly:
$S \to aAa$
$A \to Sb|bCC$
$C \to abb$ ■

4.2 Derivace a derivační stromy

Derivace jako taková byla už popsána a ukázaná na příkladech v kapitole 2.2 a bezkontextové gramatiky v kapitole 3. Proto postačí, když si jen připomeneme, že derivace slova končí, pokud po posledním kroku obsahuje derivovaný řetězec pouze terminály (terminální slovo) a že existuje levá nebo pravá derivace. Derivace, ale nemusí vždy končit úspěšně jestliže:

(a) zadaná gramatika neumí generovat žádná slova nebo

(b) zadaná gramatika neumí generovat konkrétní (hledané) slovo.

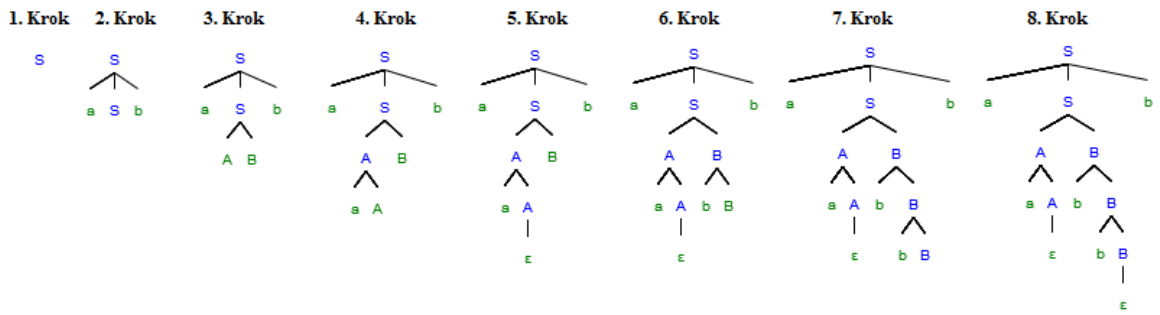
Nyní přejdeme na druhou část kapitoly a tou jsou derivační stromy. Derivační stromy jsou druhým způsobem zápisu derivace a to pomocí zakreslení (graficky). Sestrojení derivačního stromu (viz definice 3) začíná nejčastěji od počátečního neterminálu odshora a postupně pokračuje směrem dolů (viz obrázek 2). Každý neterminál, který uzel ve stromě obsahuje se rozšiřuje na další uzly s neterminály a listy s terminály tak dlouho, dokud se nepřepíši pouze na nějaký terminální symbol (list). Terminál ve stromě ukončuje větvení (je listem) pro danou větev. Jakmile derivace skončí lze vidět derivované slovo při průchodu koncových listů ve stromě ve směru zleva doprava. V případě, že existuje více levých derivací ke stejnému slovu jsou derivační stromy odlišné a jedná se o tzv. nejednoznačné (víceznačné) gramatiky. Co je to víceznačná gramatika už zaznělo v kapitole 2.2.

Definice 3 *Derivační strom je uspořádaný kořenový strom, kde*

- *vrcholy stromu jsou ohodnoceny prvky* $N \cup T \cup \{\varepsilon\}$,
- *kořen je vždy ohodnocen počátečním neterminálem,*
- *vrchol ohodnocený prvkem* X, *kde* $X \in N$ *obsahující potomky tak, že pro* $Y_1, Y_2, \ldots, Y_n$ ($Y_i \in N \cup T \cup \{\varepsilon\}$, *pro* i=1,...,n), *kde* $(X \to Y_1 Y_2 \ldots Y_n) \in P$,
- *vrchol ohodnocený* a, *kde* $a \in T$, *je listem*
- *vrchol ohodnocený* ε *je opět listem, který je jediným následníkem svého rodiče*

Pro představu je konstrukce derivačního stromu zobrazena na obrázku 2 a levá derivace k ní na příkladu 4.

Příklad 4
Vstupní gramatika G a počáteční neterminál je S, kde
$S \to AB \mid aSb$
$A \to aA \mid \varepsilon$
$B \to bB \mid \varepsilon$
Levá derivace: $S \Rightarrow aSb \Rightarrow aABb \Rightarrow aaABb \Rightarrow aaBb \Rightarrow aabBb \Rightarrow aabbBb \Rightarrow aabbb$
Otázka: Lze BG generovat slovo $w = aabbb$?
Odpověď: Ano, slovo $w = aabbb$ lze generovat zadanou BG. ■



Obrázek 2: Konstrukce derivačního stromu

Slovní postup konstrukce derivačního stromu uvedeného na obrázku 2 je:

1. Krok – začínáme vytvořením kořene stromu, který je značen počátečním neterminálem S
2. Krok – neterminál S ve stromu byl rozšířen o následníka aSb z pravidla $S \to aSb$
3. Krok – neterminál S (rodič) aSb je rozšířen o pravou stranu pravidla $S \to AB$
4. Krok – neterminál A (rodič) AB je rozšířen o pravou stranu pravidla $A \to aA$

5. Krok – neterminál A (rodič) aA je rozšířen o pravou stranu pravidla $A \to \varepsilon$ (ukončení)

6. Krok – neterminál B (rodič) AB je rozšířen o pravou stranu pravidla $B \to bB$

7. Krok – neterminál B (rodič) bB je rozšířen o pravou stranu pravidla $B \to bB$

8. Krok – neterminál B (rodič) bB je rozšířen o pravou stranu pravidla $B \to \varepsilon$ (ukončení)

V posledním kroku jsme docílili toho, že každý neterminál ve stromě má následníka obsahujícího pouze terminální symbol. Tento symbol je konečný (nemění se) a tedy konstrukce derivačního stromu je hotová. Derivované slovo je následně nalezeno průchodem stromu po koncových listech a to čtením zleva doprava.

4.3 Převod gramatiky do Chomského normální formy

Převod gramatiky do Chomského normální formy (ChNF) [6] převede všechna pravidla do jednoho z následujících tvarů:

- $A \to XY$ – pravá strana pravidla obsahuje právě dva neterminály
- $X \to a$ – pravá strana pravidla obsahuje pravě jeden terminál
- $S \to \varepsilon$ – kde na levé straně je počáteční neterminál a na pravé straně epsilon, ale v tomto případě se počáteční neterminál nesmí vyskytovat na pravé straně žádného pravidla gramatiky

U gramatiky zadané v ChNF lze pomocí derivace dojít v konečném počtu n kroků $(2n - 1)$ ke každému slovu v ní, kde n je délka řetězce (vstupního slova). Postup převodu BG do ChNF se skládá ze 4 částí:

1. Odstraní se terminály z pravých stran – pro každý terminál 'a' vyskytující se v pravidle s pravou stranou délky 2 a více se vytvoří nový neterminál X a pravidlo $X \to a$. Poté v každém pravidle s délkou pravé strany alespoň 2 se nahradí každý výskyt 'a' naším nově vytvořeným neterminálem X.

2. Zkrátí se délky pravých stran – každé pravidlo s pravou stranou délky k, kde $k > 2$ se nahradí sadou $k - 1$ nových pravidel s $k - 2$ nově přidanými pomocnými neterminály.

3. Odstraní se epsilon (ε) pravidla – výsledkem budou nová pravidla v gramatice nahrazující původní epsilon(ε) pravidla tak, že tato pravidla budou mít na pravé straně libovolný počet vypuštěných neterminálů.Postup zní následovně:

   (1) Vytvoří se množina $\mathcal{E}$={...}, která bude obsahovat všechny levé strany pravidel (neterminály), které lze přepsat na ε.

(2) Prochází se všechna pravidla v gramatice a vytváří se nová pravidla tak, že za každý výskyt neterminálu na pravé straně pravidla v množině $\mathcal{E}$ se daný neterminál vynechá a to v libovolném počtu a pořadí, ale tak, aby pokryl všechny způsoby (kombinace) bez daného neterminálu.

- V případě, že existuje tzv. počáteční neterminál, který je v množině $\mathcal{E}$ vytvoří se pravidla $S' \to S|\varepsilon$, kde S je původní počáteční neterminál a S' nově vytvořený počáteční neterminál. Po převodu je gramatika stále schopna generovat prázdné slovo.

(3) Odstraní se pravidla ve tvaru $X \to \varepsilon$, kde X je nějaký levý neterminál pravidla obsahující na pravé straně ε, ale zároveň X není počátečním neterminálem, jak bylo uvedeno na začátku této kapitoly.

4. Odstraní se jednoduchá (unit) pravidla ($A \to B$) – nahradí se jednoduchá pravidla ($A \to B$) pravidly s A na levé straně a všemi možnými pravými stranami délky alespoň 2 nebo tvořenými jediným terminálem z pravidel, které mají vlevo neterminál dosažitelný z B jen pomocí jednoduchých pravidel. Postup zní následovně:

   (1) Pro každý neterminál X se vytvoří množina D_x všech neterminálů dosažitelných z něj pouze pomocí unit pravidel.

   (2) Pro každé unit pravidlo ve tvaru $A \to B$ se projde množina D_B a pro každý neterminál v ní obsažený se vezmou všechny pravé strany jeho pravidel (s výjimkou unit pravidel) a vytvoří se nová pravidla ze získaných pravých stran, která vlevo budou mít neterminál A.

   (3) Odstraní se všechna unit pravidla ($A \to B$).

Po provedení všech kroků bude na výstupu gramatika v ChNF. Nutno podotknout, že v případě, kdy při odstraňování epsilon pravidel vznikne pravidlo $S \to \varepsilon$, kde S je počátečním neterminálem a nachází se na pravé straně pravidla v BG, je potřeba tento problém vyřešit novými pravidly $S' \to S|\varepsilon$, kde S' bude nově počátečním symbolem gramatiky. Tím se zajistí, aby gramatika byla na výstupu v ChNF a schopná po převodu generovat prázdné slovo, aby neterminál S' (nově počáteční neterminál) se nevyskytoval nikde na pravé straně pravidel.

Příklad 5 (Převod BG do ChNF)
Bezkontextová gramatika G s počátečním neterminálem S, kde
$S \to aXbX$
$X \to aY|bY|\varepsilon$
$Y \to X|c$

1. Odstranění terminálů z pravých stran pravidel délky $n > 1$

   (a) Vytvoření nových pravidel $A \to a$, $B \to b$ z důvodu pravidel $\{S \to aXbX, X \to aY, X \to bY\}$

- Pravidlo $S \to aXbX$ upraveno na $S \to AXBX$
- Pravidlo $X \to aY$ upraveno na $X \to AY$
- Pravidlo $X \to bY$ upraveno na $X \to BY$

Gramatika na výstupu:
$S \to AXBX$
$X \to AY|BY|\varepsilon$
$Y \to X|c$
$A \to a$
$B \to b$

2. Zkrácení délky pravých stran pravidel na max délku $n \leq 2$

   (a) Potřeba upravit pravidlo $S \to AXBX$

   - $S \to AX\underline{BX}$ vytvoří se nové pravidlo $C \to BX$
   - Přepis $S \to AX\underline{BX}$ na pravidlo $S \to AXC$
   - $S \to A\underline{XC}$ vytvoří se nové pravidlo $D \to XC$
   - Přepis $S \to A\underline{XC}$ na pravidlo $S \to AD$

   Gramatika na výstupu:
   $S \to AD$
   $X \to AY|BY|\varepsilon$
   $Y \to X|c$
   $A \to a$
   $B \to b$
   $C \to BX$
   $D \to XC$

3. Odstranění $A \to \varepsilon$ pravidel

   (a) Množina $\mathcal{E}=\{X, Y\}$

   - Z pravidla $C \to BX$ vznikne pravidlo $C \to B$
   - Z pravidla $D \to XC$ vznikne pravidlo $D \to C$
   - Z pravidla $X \to AY$ vznikne pravidlo $X \to A$
   - Z pravidla $X \to BY$ vznikne pravidlo $X \to B$

Gramatika na výstupu:
$S \to AD$
$X \to AY|BY|A|B$
$Y \to X|c$
$A \to a$
$B \to b$
$C \to BX|B$
$D \to XC|C$

4. Odstranění $A \to B$ (tzv. jednoduchých) pravidel

   (a) Vytvoření množin dosažitelných neterminálů pro každý neterminál pomocí unit pravidel:
   $D_S = \{S\}$
   $D_X = \{X, A, B\}$
   $D_Y = \{Y, X, A, B\}$
   $D_A = \{A\}$
   $D_B = \{B\}$
   $D_C = \{C, B\}$
   $D_D = \{D, C, B\}$

   - Pro každé jednoduché (unit) pravidlo se vytvoří nová pravidla tak, že se na pravé straně pravidla nahradí neterminál všemi pravými stranami pravidel dosažitelné množiny D_x, kde x je neterminál na pravé straně unit pravidla a jednotlivé neterminály v množině D_x jsou levými stranami pravidel.
   - Nutné je dodržovat to, že nesmí vzniknout nová pravidla, která budou opět jednoduchými pravidly a zároveň už existovat (duplicitní pravidla se přeskakují).

   Gramatika na výstupu je v ChNF:
   $S \to AD$
   $X \to AY|BY|a|b$
   $Y \to c|AY|BY|a|b$
   $A \to a$
   $B \to b$
   $C \to BX|b$
   $D \to XC|b$

■

4.4 Převod gramatiky do Greibachové normální formy

Při převodu gramatiky do Greibachové normální formy (GNF) se všechny pravidla přepisují do tvaru:

- $A \to a\alpha$ kde a je terminál a $\alpha \in N^*$ nebo
- $S \to \varepsilon$ kde S je počáteční neterminál, který se nesmí nacházet nikde na pravé straně pravidel gramatiky.

Algoritmus, který převádí gramatiku do GNF [7] a byl použit v této kapitole, požaduje, aby na vstupu byla gramatika v ChNF. Postup převodu do GNF vypadá pak následovně.

1. Převést gramatiku do ChNF, pokud tomu tak není.
2. Provést na gramatice redukci k odstranění pravidel negenerujících alespoň jeden řetězec.
3. Seřadit neterminály od počátečního neterminálu S v libovolném pořadí.
   - Zvolený způsob seřazení neterminálů má vliv na pořadí v jakém budou pravidla zpracovávána, může ovlivnit velikost výsledné gramatiky, ale všechna různá seřazení vedou nakonec k gramatice v GNF.
4. V seřazeném seznamu neterminálů N začínáme od prvního z nich procházet postupně všechna pravidla taková, která mají na levé straně stejný neterminál jako aktuálně zpracovávaný neterminál v seznamu N a to pro každý N_1 až N_n, takto:
   (a) Pro každé pravidlo zapsané ve tvaru $A_i \to A_j\gamma$ s neterminálem na levé straně pravidla stejným jako ze seznamu N musí být splněna podmínka $j > i$, kde j a i jsou pořadí neterminálů vůči seznamu N.
   (b) Pokud existuje pravidlo $A_k \to A_j\gamma$, kde podmínka je $j < k$, vytvoří se nová pravidla za pomocí provedení substituce daného neterminálu A_j za všechny jeho pravé strany pravidel a potom se odstraní původní pravidlo $A_k \to A_j\gamma$.
   (c) Počet opakování kroku (b) je maximálně $k-1$ krát k získání pravidel ve tvaru $A_k \to A_p\gamma$, $p \geq k$
   (d) A zároveň se upravují (nahrazují) pravidla s levou rekurzi, která jsou ve tvaru $A_k \to A_j$, kde $k = j$ takto:
      - Pro každé pravidla neterminálu A tak, že $A \to A\alpha_1|A\alpha_2|\ldots|A\alpha_r|\beta_1|\beta_2|...|\beta s$
      - se vytvoří nové pravidla s ještě nepoužitým neterminálem (např. Z) a pravidly
        - $Z \to \alpha_i|\alpha_i Z, 1 \leq i \leq r$ a
        - $A \to \beta_i|\beta_i Z, 1 \leq i \leq s$
        - nově vytvořený neterminál Z se vloží do seznamu neterminálů N na začátek (zpracován bude až v druhé části převodu a to, až na závěr).

5. V druhé části se procházejí pravidla $A_i \to A_j\gamma$ v pořadí $i = n-1, n-2, \ldots, 1$, a upravují se podobně jako v kroku 4.(b), tak aby se na první pozici na pravé straně pravidla objevil vždy terminál (proces probíhá i pro nově vytvořená pravidla, která nahrazují levou rekurzi u pravidel).

6. Výsledkem je gramatika v GNF.

Výstupem je gramatika, která je v GNF. Algoritmus popsaný v této kapitole, jak už bylo zmiňováno, požaduje na vstupu gramatiku v ChNF. To nám usnadňuje převod, neboť není nutné řešit odstraňování epsilon a unit pravidel. Dále není potřeba v závěru procházet všechna pravidla a kontrolovat, jestli se na jiné než první pozici nenachází nějaký terminální symbol. V opačném případě by bylo nutné nahradit terminály na pravých stranách pravidel podobným způsobem, jak tomu bylo v 1. kroku u převodu gramatiky do ChNF. Popis převodu do ChNF je v kapitole 4.3.

Příklad 6 (Převod BG do GNF)
Bezkontextová gramatika G s počátečním neterminálem S, kde
$S \to XA|BB$
$B \to b|SB$
$X \to b$
$A \to a$

1. Převod gramatiky do ChNF pokud už tomu tak není a pokud je potřeba provést redukci.
   - Gramatika na vstupu je v ChNF.
   - Redukci na gramatice není potřeba provádět.

2. Zvolit pořadí neterminálů: $\{S, X, A, B\}$ (začínat nejlépe vždy od počátečního neterminálu).
   - $S \to XA|BB$
     $X \to b$
     $A \to a$
     $B \to b|SB$

3. Substituce v případě neterminálů v pořadí $A > B$ a odstraňování levých rekurzí $A \to AB$ v pravidlech.
   (a) Odstranění pravidla $B \to \underline{S}B$ z důvodu $B > S$
      - Nová pravidla $B \to XAB|BBB$
   (b) Odstranění pravidla $B \to \underline{X}AB$ z důvodu $B > X$
      - Nové pravidlo $B \to bAB$
   (c) Odstranění pravidla $B \to \underline{B}BB$ z důvodu $B = B$ (levá rekurze)
      - Odstranění provedeme zavedením nového neterminálu „C“ a pravidel
        $C \to \ BB \mid BBC$ a tím splníme první podmínku ($Z \to \alpha_i|\alpha_i Z, 1 \leq i \leq r$) pak

- vytvoříme pravidla $B \rightarrow\ bC \mid bABC$ a tím splníme i druhou podmínku $(A\ \rightarrow\ \beta_i|\beta_i Z, 1 \leq i \leq s)$ a tím jsme nahradili levou rekurzi v pravidle $B \rightarrow BBB$.

Gramatika na výstupu:
$C \rightarrow BB|BBC$
$S \rightarrow XA|BB$
$X \rightarrow b$
$A \rightarrow a$
$B \rightarrow b|bAB|bC|bABC$

4. Převod zbývajících pravidel do GNF

   - Pravidla procházíme v sestupném pořadí podle seřazených neterminálů $\{C, S, X, A, B\}$.

   (a) Odstranění pravidla $S \rightarrow \underline{X}A$

   - Nové pravidlo $S \rightarrow bA$

   (b) Odstranění pravidla $S \rightarrow \underline{B}B$

   - Nové pravidla $S \rightarrow bB|bABB|bCB|bABCB$

   (c) Odstranění pravidla $C \rightarrow \underline{B}B$

   - Nové pravidla $C \rightarrow bB|bABB|bCB|bABCB$

   (d) Odstranění pravidla $C \rightarrow \underline{B}BC$

   - Nové pravidla $C \rightarrow bBC|bABBC|bCBC|bABCBC$

   Gramatika na výstupu je v GNF:
   $S \rightarrow bA|bB|bABB|bCB|bABCB$
   $B \rightarrow b|bAB|bC|bABC$
   $X \rightarrow b$
   $A \rightarrow a$
   $C \rightarrow bB|bABB|bCB|bABCB|bBC|bABBC|bCBC|bABCBC$

■

4.5 Cocke-Younger-Kasami algoritmus

Cocke-Younger-Kasami zkráceně CYK je algoritmus [8], který pro gramatiku v ChNF formě rozhodne s časovou složitostí $O(n^3)$ zdali lze nějaké slovo gramatikou generovat nebo ne. Jak CYK algoritmu vypadá je zobrazeno pomocí Pseudokódu (viz. Algoritmus 1). Slovní popis CYK algoritmu by vypadal následovně.

1. Na začátku se inicializuje čtvercová tabulka o velikosti vstupního slova w tak, že počet řádků a sloupců je $n = |w|$.

2. Obvykle se navíc pod tabulku nebo přidáním tzv. nultého řádku přidá prostor na zápis vstupního slova (v každé buňce jeden znak ze slova).

3. Vyplňování tabulky začíná odspoda po řádcích směrem nahoru po buňkách zleva doprava.

4. V prvním řádku tabulky obsahují jednotlivé buňky všechny levé strany pravidel, kterými lze konkrétní znak ze vstupního slova v řádku pod ním pravou stranou pravidla generovat (přímo).

5. Další řádky v tabulce se vyplňují podle podřetězce ověřovaného vstupního slova w a začíná se od 2 řádku, proto $i = 2$, kde i je číslo řádku tabulky a pokračuje se až do n, kde n je délka vstupního (hledaného) slova w.

   - Pro každou buňku $x[i, j]$ v tabulce, kde i je číslo řádku a j je číslo sloupce, je potřeba vyřešit k dvojic (podřetězců vstupního slova), kde $k < i$ a $j < |w| - i$.
   - Pro každou k-tou dvojici zjistíme její položky z tabulky následovně:
     - první položka z dvojice na pozici $[k, j]$ v tabulce
     - druhá položka z dvojice na pozici $[[i - k - 1], [j + k + 1]]$ v tabulce

6. Mezi prvními a druhými položkami se provede zřetězení jejich neterminálů.

   (1) Zjistí se, jestli tomu odpovídají některé pravé strany pravidel, a pokud ano, tak se přidají jejich levé strany pravidel do nové množiny.

   (2) V závěru se tyto všechny nové množiny sjednotí a výsledkem je množina neterminálů z levých stran pravidel, která tvoří buňku x na pozici [i, j] v tabulce.

7. Vyplňování pokračuje až do $n - \textit{tého}$ řádku tabulku, kde

   (1) s každým řešeným řádkem se zvětšuje podřetězec ověřovaného slova (začíná se na 1 a pokračuje až do n=$|w|$) a

   (2) v případě, že na posledním řádku tabulky v první buňce se vyskytuje počáteční neterminál, tak z toho pak vyplývá, že vstupní slovo lze zadanou gramatikou v ChNF generovat.

U procesu vyplňování tabulky si lze všimnout, že se vyplňuje pouze spodní půlka tabulky a to vždy po diagonálu. Výsledkem je vyplněná tabulka, která má tvar pravoúhlého trojúhelníku. Algoritmus CYK patří do tzv. dynamického programování. Je to způsob řešení problémů, kdy se větší problém rozdělí do menších částí a tím ulehčí jeho vyřešení (složitost). Jednotlivé části jsou totiž postupně řešeny a jejich výsledky použity k dalším a složitějším částem. Výhodou tohoto dynamického programování je, že vyřešené jednotlivé části daného problému jsou vypočítány jen jednou a nemusí se opakovat. Tvořený celek pak dává odpověď na celý (řešený) problém. Z takto vyplněné tabulky lze, kromě rozhodnutí, jestli lze gramatikou nějaké slovo generovat nebo ne, získat zpět i derivační strom. Pro tento způsob, je nutné si pamatovat použité pravé strany pravidel a sestavovat je opačným způsobem z počátečního neterminálu směrem dolů. Při následném sestavování derivačního stromu je potřeba dávat pozor na víceznačné gramatiky, které by mohly vést více způsoby k derivovanému slovu a tedy tvořit vícero derivačních stromů. Za zmínku stojí, že Lange & Leiß (2009) [9] představili CYK algoritmus, který zvládal i gramatiky v jiné formě než jen ChNF.

Vstup: Gramatika v ChNF, $G = \{N, T, P, S\}$ a vstupní slovo $w = a_1 a_2, \ldots, a_n \neq \varepsilon$
Výstup: Tabulka $t_{i,j}$
inicializace;
for *i=1* ***to*** n **do**
  $t_{i,1} = \{A | A \to a_i \in P\}$;
end
for *j=2* ***to*** n **do**
  for *i=1* ***to*** *n-j+1* **do**
    for *k=1* ***to*** *j-1* **do**
      $t_{i,j} = \{A | A \to BC \in P, B \in t_{i,k}, C \in t_{i+k,j-k}, 1 \leq k \leq j\}$;
    end
  end
end

Algoritmus 1: Cocke-Younger-Kasami (CYK)

Příklad 7
Bezkontextová gramatika G s počátečním neterminálem S a vstupním slovem $w = abaa$, kde
$S \to AB|SS|a$
$A \to AA|BC|a$
$B \to AB|b$
$C \to SA|b$
Otázka: Lze slovo w generovat zadanou bezkontextovou gramatikou?
Řešení: Vytvoříme tabulku o velikosti $n * n$, kde $n = |w| = 4$. A do nultého řádku (pod tabulku) vložíme vstupní slovo w. Tomu odpovídá tabulka č. 1.

Tabulka 1: CYK - Inicializace tabulky

$a_{i/j}$	1	2	3	4
4	a_{14}	a_{24}	a_{34}	a_{44}
3	a_{13}	a_{23}	a_{33}	a_{43}
2	a_{12}	a_{22}	a_{32}	a_{42}
1	a_{11}	a_{21}	a_{31}	a_{41}
w	a	b	a	a

Potom vyplníme jednotlivé buňky tabulky odspoda způsobem, jakým byl popsán CYK algoritmus v této kapitole.

1. Buňky v řádku č. 1 získáme ověřením podřetězce vstupního řetězce w o délce 1 znaku: $\{a, b, a, a\}$ a v tomto případě přímou derivaci částí vstupního slova v nultém řádku pod uvedenou tabulkou.

   (a) Řetězec a získáme pravidly $S \to a, A \to a$

   - Buňky a_{11},a_{31},a_{41} obsahují množinu levých stran, tj. $\{S, A\}$

   (b) Řetězec b získáme pravidly $B \to b, C \to b$

   - Buňku a_{21} tvoří množinu levých stran, tj. $\{B, C\}$

2. Buňky v řádku č. 2 získáme ověřením podřetězce vstupního řetězce w o délce 2 znaků: $\{ab, ba, aa\}$, kde

   (a) pro řetězec ab a $k = 1$ se první položka rovná a_{11}=$\{S, A\}$ a druhá a_{21}=$\{B, C\}$ a zřetězením vznikne množina pravých stran, tj. M=$\{SB, SC, AB, AC\}$.

   - Buňku a_{12} a $k = 1$ pak tvoří všechny levé strany pravidel, které mají pravou stranu v množině M a tedy buňka a_{12}=$\{S, B\}$.

   (b) pro řetězec ba se první položka rovná a_{21}=$\{B, C\}$ a druhá a_{31}=$\{S, A\}$ a zřetězením vznikne množina M=$\{BS, BA, CS, BA\}$.

   - Buňku a_{22} pak tvoří všechny levé strany pravidel, které mají pravou stranu v množině M a tedy buňka a_{22}=$\emptyset$ (žádná taková pravidla neexistují).

   (c) pro řetězec aa a $k = 1$ se první položka rovná a_{31}=$\{S, A\}$ a druhá a_{41}=$\{S, A\}$ a zřetězením vznikne množina M=$\{SS, SA, AS, AA\}$.

   - Buňku a_{32} pak tvoří všechny levé strany pravidel, které mají pravou stranu v množině M a tedy buňka a_{32}=$\{S, A, C\}$.

3. Buňky v řádku č. 3 získáme ověřením podřetězce vstupního řetězce w o délce 3 znaků: $\{aba, baa\}$, kde

   (a) pro řetězec aba a $k = 1$ se první položka rovná a_{11}=$\{S, A\}$ a druhá a_{22}=$\emptyset$ a zřetězením vznikne M_1=$\emptyset$.

   (b) pro řetězec aba a $k = 2$ se první položka rovná a_{12}=$\{S, B\}$ a druhá a_{31}=$\{S, A\}$ a zřetězením vznikne množina M_2=$\{SS, SA, BS, BA\}$.

   - Výsledná buňka a_{13} obsahuje všechny levé strany pravidel, které odpovídají pravým stranám výsledné množiny M=$M_1 \cup M_2$ a tedy buňka a_{13}=$\{S, C\}$.

   (a) pro řetězec baa a $k = 1$ se první položka rovná a_{21}=$\{B, C\}$ a druhá a_{32}=$\{S, A, C\}$ a zřetězením vznikne M_1=$\{BS, BA, BC, CS, CA, CC\}$.

   (b) pro řetězec baa a $k = 2$ se první položka rovná a_{22}=$\emptyset$ a druhá a_{41}=$\{S, A\}$ a zřetězením vznikne množina M_2=$\emptyset$.

   - Výsledná buňka a_{23} obsahuje všechny levé strany pravidel, které odpovídají pravým stranám výsledné množiny M=$M_1 \cup M_2$ a tedy buňka a_{23}=$\{A\}$.

4. Buňky v řádku č. 4 získáme ověřením podřetězce vstupního řetězce w o délce 4 znaků: $\{abaa\}$, kde

   (a) pro řetězec $abaa$ a $k = 1$ se první položka rovná a_{11}=$\{S, A\}$ a druhá a_{23}=$\{A\}$ a zřetězením vznikne množina M_1=$\{SA, AA\}$.

   (b) pro řetězec $abaa$ a $k = 2$ se první položka rovná a_{12}=$\{S, B\}$ a druhá a_{32}=$\{S, A, C\}$ a zřetězením vznikne množina M_2=$\{SS, SA, SC, BS, BA, BC\}$.

   (c) pro řetězec $abaa$ a $k = 3$ se první položka rovná a_{13}=$\{S, C\}$ a druhá a_{41}=$\{S, A\}$ a zřetězením vznikne množina M_3=$\{SS, SA, CS, CA\}$.

   - Výsledná buňka a_{14} obsahuje všechny levé strany pravidel, které odpovídají pravým stranám výsledné množiny M=$M_1 \cup M_2 \cup M_3$ a tedy buňka a_{14}=$\{S, A, C\}$.

Tomuto postupu odpovídá konstrukce CYK tabulky zobrazené v tabulce č. 2. Odpověď na otázku zdali lze slovo w generovat je ano, protože počáteční neterminál S se nachází v poslední vyplněné tabulce na řádku č. 4 v buňce $a_{1,4}$ a z popisu CYK algoritmu vyplývá, že slovo $w \in L(G)$ lze generovat zadanou gramatikou.

1) Řádek

i/j	1	2	3	4
4	–	–	–	–
3	–	–	–	–
2	–	–	–	–
1	S, A	B, C	S, A	S, A
w	a	b	a	a

2) Řádek

i/j	1	2	3	4
4	–	–	–	–
3	–	–	–	–
2	S, B	$\emptyset$	S, A, C	–
1	S, A	B, C	S, A	S, A
w	a	b	a	a

3) Řádek

i/j	1	2	3	4
4	–	–	–	–
3	S, C	A	–	–
2	S, B	$\emptyset$	S, A, C	–
1	S, A	B, C	S, A	S, A
w	a	b	a	a

4) Řádek

i/j	1	2	3	4
4	A, C, S	–	–	–
3	S, C	A	–	–
2	S, B	$\emptyset$	S, A, C	–
1	S, A	B, C	S, A	S, A
w	a	b	a	a

Tabulka 2: Konstrukce CYK tabulky

■

4.6 Earleyho algoritmus

Earleyho algoritmus [10] slouží k zjišťování, zdali lze nějakou bezkontextovou gramatikou generovat nějaké slovo. U algoritmu není potřeba, aby se gramatika převáděla do některé z normálních forem, jak tomu bylo např u CYK algoritmu zmíněném v kapitole 4.5. Před popisem Earleyho algoritmu, je zapotřebí si popsat co je to Earleyho položka a Earleyho množina (Set).

- Earleyho položka
  - Položka je tvořena pravidlem a číslem.
  - Číslo (index) ukazuje na pozici v pravidle, která je zpracovaná a na druhou část, která ještě zbývá zpracovat.
  - V pravidle se obvykle pozice značí symbolem tečky (•).
  - Zápis pravidla vypadá pak následovně: $X \rightarrow Y \bullet Z$, kde část pravidla před tečkou (•) je už zpracována a část za tečkou ještě zbývá zpracovat.
- Earleyho množina
  - Earleyho množina obsahuje Earleyho položky.
  - Pokud ověřujeme např. slovo w=*abba*, tak počet množin obsahujících earleyho položky bude odpovídat délce vstupního slova w a tedy délka slova $|w|$=4 +1 kvůli inicializačnímu kroku.

– Inicializační množina má v prvním kroku pouze položku s pravidlem $S' \to \bullet S$, která se pak doplní dalšími položkami pomocí metody *Predikce*.

Postup Earleyho algoritmu bude vypadat následovně takto.

1. Vytvoří se Earleyho položka s pravidlem $S' \to \bullet S$, kde S' bude nový neterminální symbol a tato položka bude sloužit k ověření na konci algoritmu, zdali lze nějaké slovo zadanou gramatikou generovat.

2. Provede se metoda *Predikce*, která přidá nové Earleyho položky s pravidly obsahujícími na levé straně neterminál S.

3. Volání metod *Predikce, Kompletace, Skener* v cyklu po n=$|w|$ kroků, kde w je vstupní slovo.

   - V každém cyklu se v další množině zpracuje další znak ze vstupního slova pomocí metody Skener.

4. Kontrola, jestli v poslední množině se nachází položka $S' \to S\bullet$ s počátečním indexem na 0.

   - Počáteční index na 0 znamená, že lze generovat vstupní slovo v celé jeho délce od počátečního indexu po délku vstupního slova w.
   - Položka existuje – slovo lze generovat zadanou gramatikou
   - Položka neexistuje – slovo nelze generovat zadanou gramatikou

Během postupu Earleyho algoritmu zazněly metody: *Predikce, Skener a Kompletace*. Nyní si tyto metody v algoritmu popíšeme:

1. Predikce

   - Pro každou položku s pravidlem ve tvaru $S \to \bullet X \ldots$, kde se tečka ($\bullet$) nachází před neterminálním symbolem se vytvoří nové položky s pravidly, které mají na levé straně neterminál X a tím vzniknou položky $X \to \bullet \ldots$, kde n je počet pravidel s levým neterminál X a tečkou ($\bullet$) na začátku.
   - Metoda zpracovává neterminální symboly.

2. Skener

   - Pro každou položku s pravidlem $S \to \ldots X \bullet a \ldots$, kde se tečka ($\bullet$) nachází před terminálním symbolem se provede porovnání s dalším očekávaným znakem ze vstupního slova w a v případě shody se tečka ($\bullet$) posune o $+1$ pozici doprava $S \to \ldots Xa \bullet \ldots$.
   - Metoda zpracovává terminální symboly.

3. Kompletace

   - Pro každou položku s pravidlem $S \to \ldots Xa \ldots \bullet$, kde se tečka ($\bullet$) nachází na konci pravidla se projdou všechny položky v množině s neterminálem S na pravých stranách a vytvoří nové položky $Z \to \ldots \bullet Sa \ldots$ tak, že se posune tečka ($\bullet$) o $+1$ pozici doprava $Z \to \ldots S \bullet a \ldots$.

Co se ale stane, když je na vstupu gramatika, která obsahuje epsilon (ε) pravidla a v derivaci jsou potřeba použít tak, aby se došlo k nějakému slovu? Earleyho algoritmus by skončil s odpovědí, že takové slovo nelze generovat, ale přitom by to nebyla pravda. Problém je způsoben tím, jak se řeší jednotlivé Earleyho položky v průběhu algoritmu, kdy se nezpracovávají ty neterminály v pravidlech, které by se jinak mohly přepsat na prázdné slovo. Nejednodušším řešením by bylo odstranit všechna taková epsilon pravidla v gramatice, tak jak bylo popsané v kapitole 4.3, aby byl přitom zachován jazyk. Jiným řešením by bylo přidat podporu pro epsilon pravidla (vypouštějící gramatiku). Jeden ze způsobu popsal John Aycock a R. Nigel Horspool [11]. Řešení spočívá v tom, že si vytvoříme např. množinu E a naplníme ji všemi neterminály, které lze přepsat na prázdné slovo (podobně jako při odstraňování epsilon pravidel v převodu BG do ChNF). Potom při každém volání metody *Predikce*, kdy se přidávají nové položky do množiny, zkontrolujeme v položce další symbol na pravé straně pravidla za tečkou ($\bullet$). V případě, že tento symbol, který se nachází za tečkou ($\bullet$), je neterminální symbol a nachází se v množině E, tak se přidá nová položka s tímto pravidlem a posunutou tečkou ($\bullet$) na pravé straně pravidla o $+1$ pozici doprava. Tím se vyvarujeme špatnému vyhodnocení u gramatik s epsilon pravidly.

Příklad 8 (Earleyho algoritmus)
Bezkontextová gramatika G s počátečním neterminálem S a vstupní slovo $w = a * a + a$, kde
$S \to S + A | A$
$A \to A * B | B$
$B \to (S) | a$
Otázka: Lze slovo w generovat zadanou bezkontextovou gramatikou?
Řešení:

1. Vytvoření pomocné položky $[Z \to S]$ hrající roli $S' \to S$
2. Set(0) – Inicializační earlyho množina
   (1) $[Z \to \bullet S]$ pomocné pravidlo
   (2) $[S \to \bullet S + A]$ predikce z (1)
   (3) $[S \to \bullet A]$ predikce z (1)
   (4) $[A \to \bullet A * B]$ predikce z (3)
   (5) $[A \to \bullet B]$ predikce z (3)

(6) $[B \to \bullet(S)]$ predikce z (5)

(7) $[B \to \bullet a]$ predikce z (5)

3. Set(1) a zpracovaná část řetězce $w = a \bullet *a + a$ (Symbol: a)

   (1) $[B \to a\bullet]$ skener Set(0)(7)

   (2) $[A \to B\bullet]$ kompletace Set(0)(5)

   (3) $[A \to A \bullet *B]$ kompletace Set(0)(4)

   (4) $[S \to A\bullet]$ kompletace Set(0)(3)

   (5) $[S \to S \bullet +A]$ kompletace Set(0)(2)

   (6) $[Z \to S\bullet]$ kompletace Set(0)(1)

4. Set(2) a zpracovaná část řetězce $w = a * \bullet a + a$ (Symbol: $*$)

   (1) $[A \to A * \bullet B]$ skener Set(1)(3)

   (2) $[B \to \bullet(S)]$ predikce z (1)

   (3) $[B \to \bullet a]$ predikce z (1)

5. Set(3) a zpracovaná část řetězce $w = a * a \bullet +a$ (Symbol: a)

   (1) $[B \to a\bullet]$ skener Set(2)(3)

   (2) $[A \to A * B\bullet]$ kompletace Set(2)(1)

   (3) $[S \to A\bullet]$ kompletace Set(0)(3)

   (4) $[A \to A \bullet *B]$ kompletace Set(0)(4)

   (5) $[S \to S \bullet +A]$ kompletace Set(0)(2)

   (6) $[Z \to S\bullet]$ kompletace Set(0)(1)

6. Set(4) a zpracovaná část řetězce $w = a * a + \bullet a$ (Symbol: $+$)

   (1) $[S \to S + \bullet A]$ skener Set(3)(5)

   (2) $[A \to \bullet A * B]$ predikce z (1)

   (3) $[A \to \bullet B]$ predikce z (1)

   (4) $[B \to \bullet(S)]$ predikce z (3)

   (5) $[B \to \bullet a]$ predikce z (3)

7. Set(5) a zpracovaná část řetězce $w = a * a + a\bullet$ (Symbol: a)

   (1) $[B \to a\bullet]$ skener Set(4)(5)

   (2) $[A \to B\bullet]$ kompletace Set(4)(3)

   (3) $[A \to A \bullet *B]$ kompletace Set(4)(2)

   (4) $[S \to S + A\bullet]$ kompletace Set(4)(1)

   (5) $[S \to S \bullet +A]$ kompletace Set(0)(2)

   (6) $[Z \to S\bullet]$ kompletace Set(0)(1)

Je vidět, že poslední Set(5) obsahuje pomocné pravidlo $Z \to S$. Toto pravidlo bylo celé zpracováno (tečka $\bullet$ se nachází na konci pravidla) a je zde odkaz na Set(0) - začátek. Víme tedy, že vstupní slovo $w = a * a + a$ lze v celé své délce vygenerovat pomocí této zadané gramatiky. ■

5 Analýza

5.1 Realizace webové aplikace

Cílem této práce jsou výukové stránky (webová aplikace), které na vstupu dostanou bezkontextové gramatiky v textovém formátu a které tyto BG zpracují. Tyto bezkontextové gramatiky slouží jako zadání (příklady) a jako vstup do implementovaných algoritmů. Algoritmy zadané BG vyřeší a na výstupu vrátí (zobrazí) řešení a postupy, jakým je algoritmy řešili a vyřešili. Kromě tohoto způsobu, kdy by si uživatelé zadali vlastní bezkontextovou gramatiku na vstupu, bude možné si vybrat již předpřipravené příklady (alespoň 5 pro každý algoritmus) a tím umožnit uživateli na výukových stránkách si zobrazit postup bez nutnosti zadávat u každého algoritmu vlastní BG na vstupu. Předpřipravené příklady bude možné upravovat a přidávat podle potřeby a to na straně serveru (webové aplikace) a to tak, aby nebylo potřeba upravovat a sestavovat celou webovou aplikaci znova. Tím, čím bude webová aplikace disponovat navíc bude možnost vygenerování bezkontextové gramatiky. Bezkontextová gramatika bude tvořena podle zadaného počtu neterminálů, terminálů atd. Výstupem webové aplikace budou postupy a řešení příkladů. Algoritmy starající se o vyřešení příkladů budou uživatelům na výstupu zobrazovat postupy, jakým se příklady řešily, aby se mohli tento proces naučit a zopakovat, např. během studia na probírané látce. Navíc pro správce stránek (aplikace) bylo umožněno upravování zobrazovaných textů u algoritmů zobrazujících postup a tím podle potřeby tyto texty v budoucnu měnit, ať už např. z důvodu opravy překlepů (chyb) nebo lépe vystihnout daný problém (převod). Další možností by bylo překládání do jiných jazyků, ale tam by bylo potřeba do-implementovat možnost přepínání mezi jednotlivými jazyky a navíc někoho, kdo dané texty přeloží. Tato funkce není náplní implementace a proto nebyla ani do realizace aplikace zahrnuta. Závěrem této kapitoly si představíme takzvanou vizi webové aplikace (výukové stránky), která vypadá následovně:

- Co – Výukové stránky, které budou mít za cíl přiblížit základy bezkontextové gramatiky.
- Jak – Množství algoritmů k vyzkoušení a k zobrazení postupu řešení (CYK algoritmus, redukce, derivace, převod do ChNF a GNF).
- Kde – Přistup možný z PC nebo mobilního zařízení na výukové stránky bez nutnosti potřeby se přihlašovat.
- Kdo – Přistupovat na výukové stránky mohou, jak studenti tak i běžní uživatelé (návštěvníci).
- Kdy – Při potřebě pochopit základy některých z implementovaných algoritmů nebo ověření znalostí a zobrazení postupu řešení a nebo jen zcela čistě ze zvědavosti a jiných důvodů nevyjímaje.

- Proč – Potřebujeme výukové stránky, které by umožnily studentům si vyzkoušet algoritmy řešící BG a u kterých by si mohly zobrazit postup a snadněji, tak pochopit základy probírané látky u bezkontextových gramatik.

6 Návrh

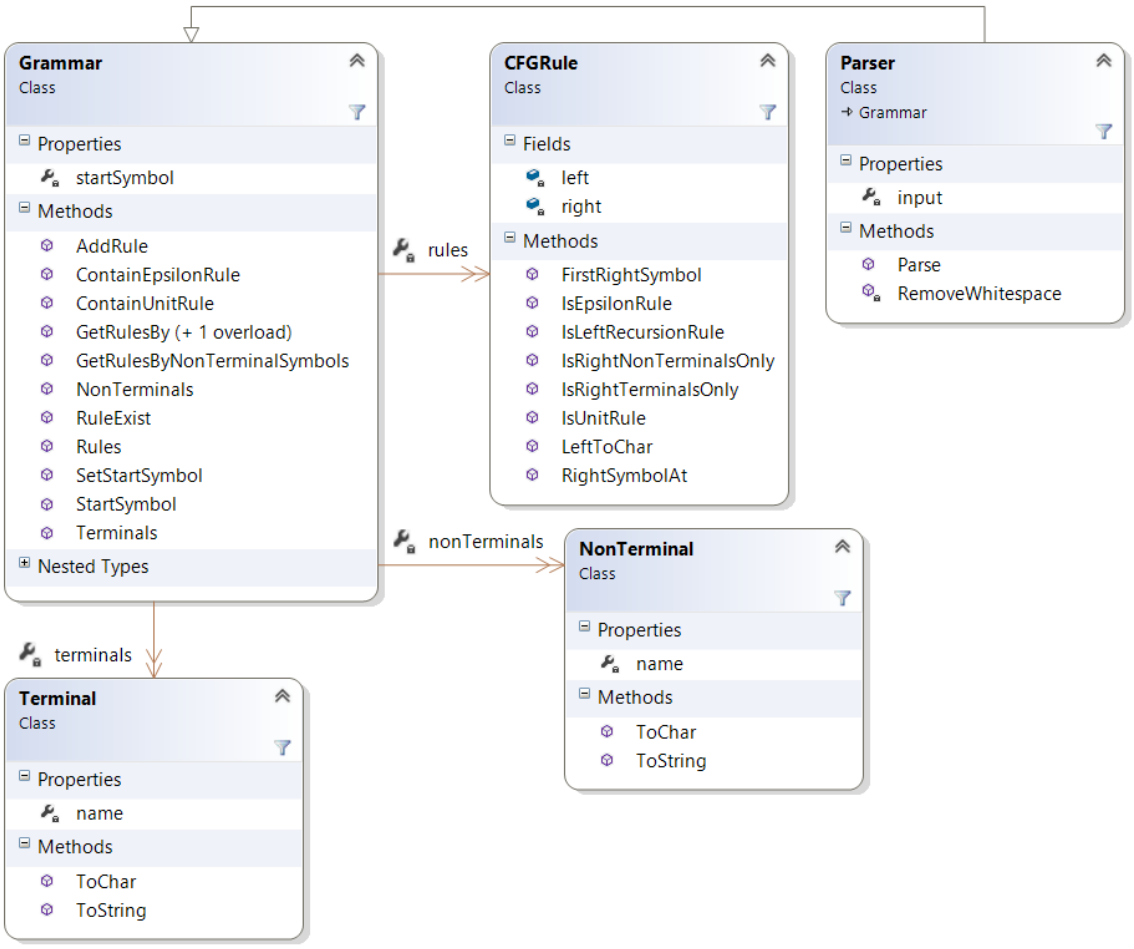
Cílem této práce jsou výukové stránky, jak už bylo zmíněno v minulých kapitolách, stránky jsou zaměřeny na seznámení studentů s bezkontextovými gramatikami. Hlavními body (funkcemi) webové aplikace budou:

1. Ukládání bezkontextových gramatik (tzv. BG) do souborů (.txt) a načítání jich zpět na stránky k dalšímu použití.
2. Generování BG podle počtu neterminálů, terminálů, pravidel a typu gramatiky (CFG, ChNF, GNF).
3. Algoritmus CYK k ověření, jestli lze gramatikou v ChNF generovat nějaké zadané slovo.
4. Uživatelsky řízenou derivaci podle zvoleného typu derivace (levá, pravá).
5. Zobrazení derivace zadaného slova v zadané gramatice (Earleyho algoritmus).
6. Zobrazení derivačního stromu (JavaSript doplněk syntree[2]) a metody k jeho naplnění (vykreslení).
7. Algoritmus pro redukování BG.
8. Algoritmy pro převod BG do ChNF a GNF.
9. Testování redukce BG a odstraňování epsilon a unit pravidel na předpřipravených příkladech.
10. Upravování předpřipravených příkladů zobrazovaných na stránce (na straně serveru).
11. Rozvržení zátěže na systém mezi klienta (JavaScript, Cookies, ...) a serveru (ASP.NET).
12. Validace zadaných slov a zadaných pravidel na straně klienta (prohlížeče).
13. Podpora multiplatformnosti mezi prohlížeči a zařízeními na kterých se stránka bude zobrazovat a to za pomocí frameworku Bootstrap[3]).
14. Snadné upravování textů zobrazovaných na stránce u algoritmů pomocí tzv. **.resx** (Resource) souborů, kde bude potřeba daný projekt znovu sestavit, aby se změny projevily (pro editování textů bude sloužit editor, který je součástí Visual Studia).

[2] `https://github.com/mshang/syntree/wiki`
[3] `https://getbootstrap.com/`

6.1 Reprezentace bezkontextové gramatiky

Na obrázku 3 je zobrazen třídní diagram popisující bezkontextovou gramatiku. Tento třídní diagram po přepsání do programovacího jazyka (kódu) bude tvořit součást webové aplikace. Hlavním úkolem je zpracovat bezkontextovou gramatiku na vstupu a ukládat jí do třídy Grammar. Jakmile BG bude uložena do třídy (objektu) Grammar, bude možné používat její metody v ní uvedené: přidat pravidlo, vypsat pravidla, změnit počáteční neterminál BG, atd.



Obrázek 3: Třídní diagram – CFG

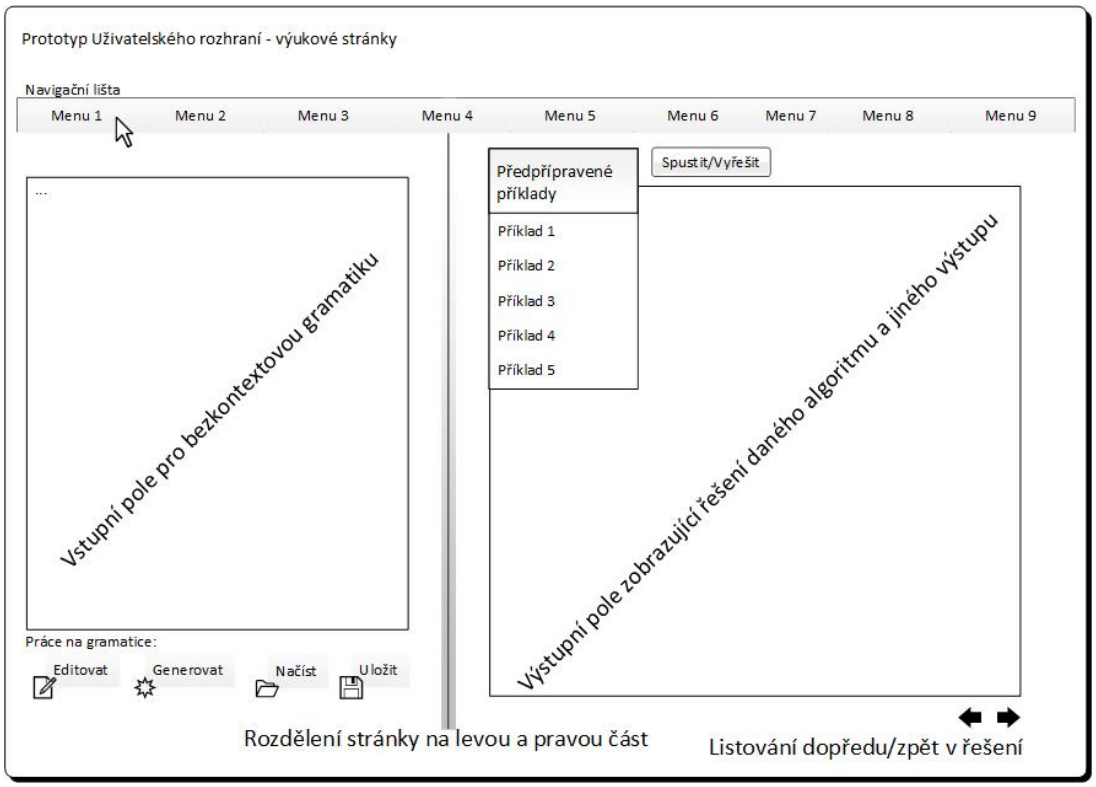
6.2 Reprezentace algoritmů bezkontextové gramatiky

Na obrázku 4 je zobrazen třídní díagram Context-Free Grammar Algorithms, zkráceně CFG_Algorithms reprezentující algoritmy uvedené v kapitole 4. Tyto algoritmy (třídy): CYK, EarleyParser, atd. v třídním diagramu potřebují na vstupu bezkontextovou gramatiku k běhu (provedení). K tomu slouží již dříve uvedený objekt (třída) Grammar z třídního diagramu CFG uvedeného v kapitole 6.1 a tak jako třídní diagram CFG se i tento přepíše do programovacího jazyka c# a bude tvořit součást webové aplikace.

Obrázek 4: Třídní diagram – CFGAlgorithms

6.3 Grafického uživatelského rozhraní

Návrh grafického uživatelského rozhraní webové aplikace (výukové stránky) je zobrazen na obrázku 5. Toto grafické uživatelské rozhraní, zkráceně GUI bylo navrženo tak, aby usnadnilo a zpříjemnilo pobyt návštěvníků na nich a to po celou dobu jejich používání. V kapitole 7.3 je pak možné vidět, jak bylo toto navrhované GUI použito ve webové aplikaci.



Obrázek 5: Návrh uživatelské rozhraní webové aplikace

7 Implementace

Program je implementován v programovacím jazyce C#. Samotná webová aplikace pak konkrétně v jazyce ASP.NET Web Forms. Při použití nebo otestování BG na stránce se BG ukládají do Cookies a to do doby, než uživatel sám ukončí webový prohlížeč nebo ho prohlížeč sám po nějaké době nesmaže. Bezkontextové gramatiky jsou v programu přístupné pomocí třídy Grammar a knihovny CFG (.dll). V následujících ukázce bude ukázáno, co se dá s touto třídou (knihovnou) dělat (viz výpis 1).

```
// zapis BG do retezce inputGrammar
string inputGrammar = "S -> aS | ASA | AB\nA -> a | B\nB -> bB | b";
Grammar grammar = new Parser(inputGrammar).Parse();
// pridani noveho pravidla zvlast
grammar.AddRule(new CFGRule(new NonTerminal('T'), ...));
// vypis vsech pravidel BG s pocatecnim neterminalem na leve strane
foreach (CFGRule rule in grammar.GetRulesBy(grammar.StartSymbol()))
{
   Console.WriteLine(rule);
}
// kontrola BG na pritomnost epsilon a unit pravidel
if (grammar.ContainEpsilonRule() && grammar.ContainUnitRule())
   Console.WriteLine("BG obsahuje epsilon a unit pravidla.");
```

Výpis 1: Ukázka načtení BG za pomocí implementované knihovny CFG v jazyce C#

Je tedy možné načíst BG a provádět na ní různé operace. Další část tvoří knihovna CFGAlgorithms (.dll), která obsahuje třídy s implementovanými algoritmy, které tyto BG používají a vracejí na výstupu zpět upravené (po průběhu algoritmu). Kromě upravené BG jsou schopné zobrazit po provedení algoritmu, které pravidlo bylo z kterého vytvořeno (změněno, smazáno a atd.). V další ukázce kódu je zobrazeno použití algoritmu převádějícího BG do ChNF pomocí třídy (objektu) ChomskyNF (viz výpis 2).

```
ChomskyNF chomsky = new ChomskyNF(grammar); // Grammar grammar = ...
if (!chomsky.IsInCHNF()) // prevest jen v pripade, ze BG neni v ChNF
{
   chomsky.ConvertToChNF(); // prevedeni BG do ChNF
   if (chomsky.IsInCHNF()) { Console.WriteLine("BG je v ChNF."); }
   else { return; }
   // vypis pravidel upravene BG, ktera je nyni v ChNF
   foreach (CFGRule rule in grammar.Rules()) { Console.WriteLine(rule); }
}
```

Výpis 2: Ukázka převodu BG do ChNF v jazyce C#

Další z implementovaných algoritmů se stará také o převod BG, ale tentokrát do GNF pomocí třídy (objektu) GreibachNF(viz výpis 3).

```
Grammar grammar = ...
GreibachNF greibach = new GreibachNF(grammar);
if (!greibach.IsInGNF())
{
   greibach.ConvertToGNF(); // prevod do GNF
   // vypis pravidel, ale nyni jiz v GNF
   foreach (CFGRule rule in grammar.Rules()) { Console.WriteLine(rule); }
   // ziskani neterminalu v poradi ve kterem byly serazeny
   NonTerminal[] nonTerminalOrder = greibach.GetNonTerminalsOrder();
}
```

Výpis 3: Ukázka převodu BG do GNF v jazyce C#

V další ukázce je použit algoritmus pro redukování BG za pomocí třídy (objektu) Reduction a je opět pouze na pár řádků (viz výpis 4).

```
Grammar grammar = ...
Reduction reduction = new Reduction(grammar);
if (!reduction.IsGrammarAlreadyReduced()) // je nutne BG redukovat?
{
   reduction.Start(); // provede redukci na BG
   // informace o mnozinach T a D (ktere neterminaly jsou v mnozinach)
   List<ReductionRule> setT = reduction.SetT();
   List<ReductionRule> setD = reduction.SetD();
   // vypis pravidel, ale nyni jiz v redukovane BG
   foreach (CFGRule rule in grammar.Rules()) { Console.WriteLine(rule); }
}
```

Výpis 4: Ukázka redukce na BG v jazyce C#

V předposlední ukázce je implementovaný algoritmus CYK a tak, jak se algoritmus nazývá, tak se i jeho třída jmenuje (viz výpis 5).

```
Grammar grammar = ...
ChomskyNF chomsky = new ChomskyNF(grammar);
// kontrola a prevod BG do ChNF, aby bylo mozne pouzit CYK algoritmus
if (!chomsky.IsInCHNF(grammar.Rules())) chomsky.ConvertToChNF();
// BG a zadane slovo
```

```
CYK cyk = new CYK(grammar, "abbb");
bool exist = cyk.Parse();
if (exist) Console.WriteLine("Slovo lze zadanou BG generovat.");
// vypis tabulky CYK
string[,] cykTable = cyk.CYKTable(); // naplnena tabulka CYK
string row = string.Empty;
for (int i = word.Length; i >= 0; i--)
{
   for (int j = 0; j < word.Length - i; j++)
   {
      string cell = cykTable[i, j];
      if (String.IsNullOrEmpty(cell)) cell = string.Empty;

      if (String.IsNullOrEmpty(row)) row = cell;
      else row += "\t".PadRight(5) + cell;
   }
   Console.WriteLine(row);
   row = string.Empty;
}
```

Výpis 5: Ukázka použití CYK algoritmu v jazyce C#

Posledním z algoritmů, který si ukážeme, je Earleyho algoritmus a k jeho použití je zapotřebí použít třídu EarleyParser (viz výpis 6).

```
Grammar grammar = ...
EarleyParser earley = new EarleyParser(grammar, "abbb");
earley.Parse();
// vypis pravidel, ktere pravou derivaci dojdou k zadanemu slovu
List<CFGRule> derivationRules = earley.DerivationRules();
foreach (CFGRule rule in derivationRules){ Console.WriteLine(rule); }
// vraci retezec, ktery je pomoci synTree vykreslen do derivacniho stromu
Console.WriteLine(earley.GetDerivationTreeFromEarley(derivationRules, grammar.
    StartSymbol())); // "[S[S[S[C[a]][S[b]]][A[b]]][A[b]]]"
```

Výpis 6: Ukázka použití Earleyho algoritmu v jazyce C#

V každé z uvedených ukázek bylo zapotřebí přidat reference na knihovny CFG a CFGAlgorithms, aby bylo možné volat třídy Grammar, ChomskyNF a atd. Algoritmy v ukázkách ukazují, jak jdou lehce použít. Je úplně jedno, jestli se jedná o aplikaci napsanou v konzoli (Console Application), WinForm, Web Forms, MVC nebo jiného typu. Webová aplikace totiž používá, jak už bylo řečeno, k tomu knihovnu CFG_ServiceLayer (.dll), která se stará o výstup z algoritmů,

který převádí do HTML kódů a webová aplikace ho buď rovnou zobrazí nebo ještě dodatečně upraví před zobrazením. Kapitola č. 9 popisuje, jak jsou texty v postupech algoritmů řešeny a jak by je bylo možné dodatečně ještě změnit. V dalších kapitolách se pak lze dočíst o použitých technologiích při implementaci v kapitole 7.2. Struktuře projektu v kapitole 7.1. Rozložení podstránek na stránce v kapitole 7.3. O systémových požadavcích v kapitole 7.4. A o sestavení webové aplikace v kapitole 7.5 a nasazení webové aplikace na server v kapitole 7.6.

7.1 Struktura webové aplikace

Struktura webové aplikace je tvořena řešením (Solution) ve Visual Studiu a 4 jeho projektů (Projects). Řešení tvoří 1 hlavní projekt a 3 jeho menší projekty. Hlavním projektem je CFG_WebApp tvořící webovou aplikaci. Webová aplikace je napsána v C# ASP.NET Web Forms. Zbylé projekty jsou napsané také v C#, ale jako DLL (Dynamic-link library) knihovny. Tyto zbylé projekty se nazývají: CFG, CFG_Algorithms a CFG_ServiceLayer. To co jednotlivé projekty dělají je popsané zde:

1. CFG
    - Projekt obsahuje třídy: Grammar, Parser, CFGRule, Terminal, NonTerminal a tyto třídy slouží pro práci s BG.
2. CFG_Algorithms
    - Projekt obsahuje třídy: Reduction, Derivation, ChomskyNF, GreibachNF, EarleyParser, CYK a jejich pomocné třídy.
    - CFG_Algorithms využívá metody a třídy projektu CFG.
3. CFG_ServiceLayer
    - Projekt obsahuje třídy: ReductionModel, DerivationModel, ChNFModel, GNFModel, EarleyModel, CYKModel, GenerateGrammarModel, Check_GNFModel, Check_ReductionModel, ChNFEpsilonRuleModel, ChNFUnitRuleModel.
    - CFG_ServiceLayer využívá metody a třídy projektů CFG a CFG_Algorithms k provedení algoritmů na BG a kde výstupem jsou řešení BG s postupy v HTML kódu.
4. CFG_WebApp
    - Projekt tvoří jednotlivé podstránky webové aplikace se scripty a moduly jako jsou JS, jQuery, CSS a Bootstrap.
    - CFG_WebApp využívá metody a třídy projektu CFG_ServiceLayer.

Zvolená struktura řešení rozděluje problém (téma diplomové práce) do jeho menších části.

7.2 Použité technologie na tvorbu webové aplikace

V této kapitole budou uvedeny jednotlivé technologie, které byly použity na tvorbu webové aplikace v průběhu implementace. Pro přehlednost jsou uvedeny v bodech a jedná se vždy o technologii, které hrála nějakou větší roli při tvorbě výukových stránek, mezi tyto technologie patří:

1. ASP.NET Web Forms [12] – pro tvorbu webové aplikace a jedná se o to, k čemu uživatelé budou přistupovat pomocí internetového prohlížeče.

2. JavaScript [14], jQuery [15] – programovací kód (scripty), které se spouští převážně na straně klienta.

3. Bootstrap [13] – starající se o vzhled stránky (tlačítka, formuláře, ...) a zobrazování (rozložení) stránek na různě velkých monitorech podporující multiplatformnost (podporu mezi vícero internetovými prohlížeči a zařízeními).

4. Cookies [16] – pro ukládání dat (informací) na straně klienta pro pozdější potřebu a kde doba uchovávání těchto dat se liší v závislosti na nastavení (hodina, den, po uzavření prohlížeče, atd.).

5. Modul synTree [17] – pro vykreslování derivačního stromu u derivace (autorem je Miles Shang).

6. JavaScript modul vex [18] – použity pro vysouvací (vyjíždějící) menu z levého rohu obrazovky u algoritmu CYK.

7. jQuery modul Steps [19] – použit pro zobrazení postupu u algoritmů převádějících BG do ChNF, GNF a algoritmu pro redukci BG (autorem doplňku je *Rafael Staib*).

Programovací jazyk ASP.NET Web Forms jsem zvolil z důvodu podobnosti (základních vlastností) s programovacím jazykem C# Windows Forms. Ostatní zvolené technologie byly pak zvoleny na základě snadné použitelnosti (využitelnosti) ve webové aplikaci nebo z důvodu toho, že jsem nevěděl o lepším řešení, ale pro potřeby dané aplikace stačily. Na začátku bylo tedy potřeba vybrat programovací jazyk v jakém bude webová aplikace napsána a rozhodnout, jak řešit některé situace a to např. jak ukládat data ze stránek, jestli na straně serveru (Session) nebo na straně klienta (Cookies), jestli použít už nějaká hotová řešení (existující způsoby, doplňky, moduly) pro zobrazení obrázků (derivačních stromů), dialogů, formulářů a atd. Bylo totiž nejednou při implementaci zjištěno, že některé vybrané technologie a způsoby řešení se budou muset řešit jinak nebo nahradit jinou technologii. Důvod byl např. ten, že server, kde stránky běžely a byly testovány, měl omezenou velikost paměti RAM, která se dala využít a stránky běžely ve společném poolu (prostoru). Na tomto sdíleném prostoru byly i stránky vícero zákazníků (lidí) a v případě problémů byl ovlivněn i zbytek skupiny. Proto např. při restartu nebo pádu

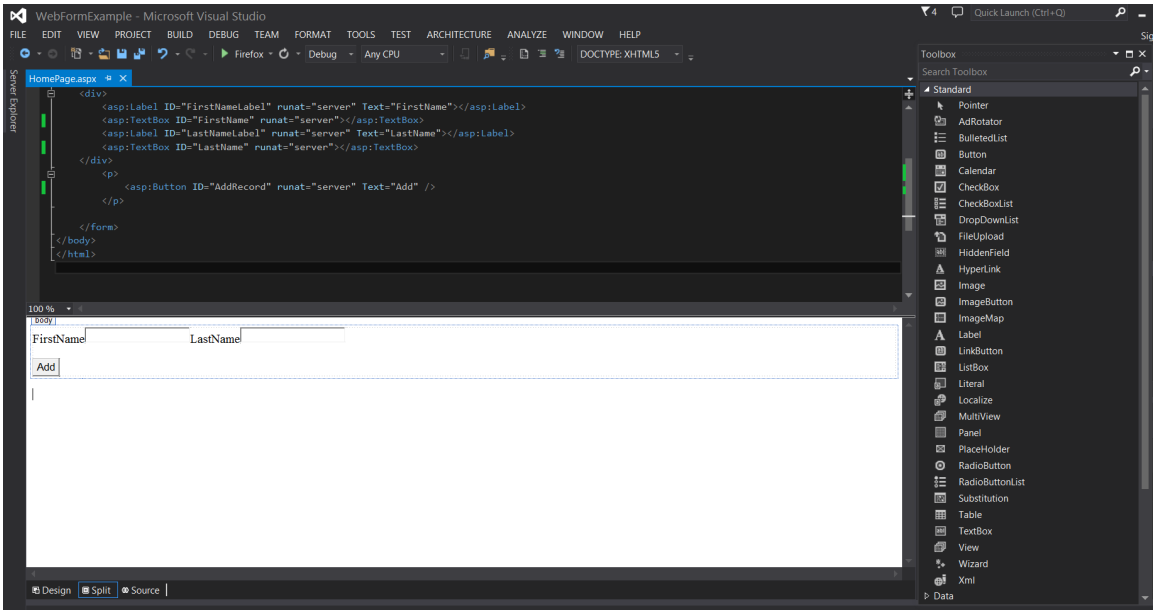
byla data na straně serveru vymazána (ztracena). To vedlo k tomu, že stránka po aktualizaci zobrazovala prázdný obsah namísto už zapamatovaných dat (vstupu) z minula. Omezení byla i další, ale jenom toto bylo pro mě problémové, kromě toho jsem s danou službou byl a jsem spokojen. Proto ukládání dat a jiných informací na straně serveru nepadalo v úvahu. Ukládání dat při používaných algoritmech na stránce se muselo změnit a to tak, aby se ukládalo na straně uživatele a to v PC pomocí Cookies namísto původního použití Session na serveru. Dalším problémem byl původně zamýšlený způsob zobrazování uživatelsky řízené derivace, kde průběh byl ukládán také na pozadí stránky ve skrytém pomocném textovém poli a bylo ho proto potřeba přepsat a začít používat Cookies. V neposlední řadě zobrazení tabulky u CYK algoritmu. Dané zobrazení bylo upraveno tak, aby data byla uložena v prohlížeči a při kliknutí na tabulku (políčko v tabulce) byl zavolán JavaScript, který zobrazí příslušný text a v tabulce označí příslušné okolní políčka ta, která s nimi souvisí. Jedná se tedy o interakci, která probíhá na straně klienta bez nutnosti posílat dotazy na server a čekat na odpověď. Výhodou tohoto přístupu bylo rozložení zátěže mezi serverem a klientem, tím se taky částečně snížila závislost na připojení k serveru a nutnosti spoléhat na to, že při brouzdání na stránkách server aplikaci nerestartuje nebo neukončí a nepřijde se tím o data. Pokud se algoritmus jednou zavolá a příklad vyřeší, je možné na dané stránce nerušeně tento příklad prohlížet. Výjimkou je algoritmus použitý u uživatelsky řízené derivace, který při vybrání pravidla komunikuje se serverem a aktualizuje zobrazený průběh derivace v dalším kroku na základě odpovědi serveru, ale nemělo by se stát, že by se přišlo o probíhající derivaci pokud by se nezavřel internetový prohlížeč a tím se nezmazaly Cookies uchovávající informace o derivaci. Na závěr kapitoly je zde krátký popis v technologií ASP.NET Web Form, JavaScript a Bootstrap.

7.2.1 ASP.NET Web Forms

ASP.NET Web Forms [12] je součástí Microsoft Visual Studia a je to vývojové prostředí, s kterým lze programovat a vytvářet různé aplikace (služby, knihovny a jiné). Daný software je od společnosti Microsoft a obsahuje programovací jazyky, jako jsou: c, c++, c#, visual basic, ale podporuje také např. JavaScript, HTML, CSS pro vývoj stránek. Samozřejmostí je ASP.NET Web Forms, který umožňuje vytvářet webové aplikace (stránky) bez nutnosti psát stránky takříkajíc od nuly. Na obrázku 6 je vidět na pravé straně tzv. „Toolbox“, kterým lze za pomocí kurzoru myší vybírat jednotlivé položky, které lze přetáhnout do spodní strany stránky myší a tím se stanou její součástí. V horní části stránky pak lze vidět „HTML“ kód, kterým lze upravit tento přidaný prvek.

7.2.2 JavaScript

JavaScript, zkráceně (tzv. JS) je programovací jazyk [14] známý tím, že běží většinou na straně klienta v prohlížeči (Firefox, Chrome, IE, ...). Je velká pravděpodobnost, že se s ním alespoň každý, kdo používá internet a brouzdá po internetových stránkách už setkal. A přitom běžný



Obrázek 6: Visual Studio – ASP.NET Web Forms

uživatel ani nemusí vědět, že stránka tento programovací jazyk (scripty) používá. A to, až do doby, než se např. zobrazí hláška, že je potřeba pro správné fungování JavaScript povolit. Ve výchozím stavu ho totiž může mít internetový prohlížeč vypnutý. Může se stát, že script se snaží spustit nějaký nebezpečný kód a v tom případě zasáhnout a zobrazit vyskakující hlášku o scriptu, který se snaží provést nějakou nebezpečnou akci. V dalším případě se může jednat o script, který přestane odpovídat (zacyklí se) a prohlížeč vyzve uživatele k tomu, jestli má daný script přerušit, nebo se jej snažit nechat doběhnout dokonce. Vyskytuje se tedy alespoň v nějaké určité míře na stránce, která není jen čistě statickou, ale dokáže podle okolností reagovat a je tedy dynamická. Rozdíl mezi dynamickou a statickou stránkou spočívá v tom, že dynamická na rozdíl od statické dokáže zareagovat a měnit obsah stránek podle událostí a např. toho, co do něj uživatel zadává a odesílá na server (potvrzuje). Může se jednat o změnu provedenou na straně serveru a poslanou zpět, aby vykreslila na stránce jiný obsah. V případě JavaScriptu je možné akce provádět na straně klienta, a to v prohlížeči. Jedná se o většinou o reakce na stisknutí tlačítka na stránce nebo jakoukoliv interakci bez nutnosti posílat cokoliv na server a čekat na jeho odpověď.

1. Výhody
    - Operace se provádí na straně klienta a tím se zatížení přesouvá ze serveru na klienta.
    - JavaScript lze použít v prohlížeči v PC, tabletu, mobilu a atd. (multiplatformní).
2. Nevýhody
    - Zdrojové kódy jsou přístupné (lze přečíst) na straně klienta.

- Kladou výkonnostní nároky na stranu klienta pokud obsahují složité výpočty (operace).
- JavaScript lze přepsat nebo upravit před odesláním s dotazem na server (je potřeba kontrolovat, co se vrací).
- JavaScript lze u klienta vypnout a tím znemožnit spuštění a např. nevykonat potřebnou akci k vykreslení obrázku a atd. proto není dobré spoléhat 100% na tento způsob.

7.2.3 Bootstrap

Bootstrap je technologie [13] tvořena JS a jQuery scripty (pluginy) a díky které lze použít již vytvořené styly (vzhledy) k zobrazení jednotlivých tlačítek, seznamů, formulářů, menu a atd. na stránce a kde je počítáno s tím, jak se má stránka zachovat v případě, kdy je zobrazována na monitoru na PC nebo na mobilu (tabletu, ...) a atd. pomocí tzv. *Grid* systému. Autory jsou Mark Otto a Jacob Thornton a byl původně vytvořen pro Twitter[4]. První verze byla vydána v roce 2011 a od té doby je známým a hojně používaným doplňkem při vytváření stránek. Jedná se o již vytvořené pravidla, styly, funkce, které se udržují, opravují, zlepšují a není je potřeba opětovně znovu vymýšlet a zapisovat. Výhodou je, že stránky se vždy zobrazí tak, aby byl všechen obsah vidět i za předpokladu, že by bylo nutné obsah přeuspořádat (změnit) na stránce tak, aby se zobrazil i na menším monitoru (rozlišení).

7.3 Rozvržení webové aplikace

Rozvržení webové aplikace popisuje jednotlivé podstránky, které tvoří webovou aplikaci (výukové stránky). Návštěvou těchto stránek pomocí webového prohlížeče, ať už na PC, tabletu nebo mobilu se zobrazí „Úvodní stránka“, která je tzv. domovskou stránkou. Pro následnou navigaci na stránce pak slouží navigační lišta, která je umístěna vždy nahoře na stránce pro snadnou navigaci po ní. Tato navigační lišta má za cíl usnadnit přístup k jednotlivým podstránkám, ať už obsahující algoritmy a nebo jiné podstránky. Dále každá stránka je rozdělena na levou a pravou stranu, kde pouze pravá strana se mění v závislosti na podstránce, kterou si uživatel zvolí např. z navigační lišty. Tohoto bylo docíleno za pomocí tzv. MasterPage v ASP.NET projektu. MasterPage obsahuje navigační lištu a pevně zadané rozložení stránek na levou a pravou stranu. Levá strana stránky obsahuje vždy vstupní pole pro zápis BG uživatelem a také zobrazení aktuálně načtené BG. MasterPage tedy zjednodušeně zobrazí vždy ten samý obsah u podstránek, které si jej u sebe nadefinují (odkážou na MasterPage). Nedochází tedy k redundanci kódu a navíc díky tomu v tomto případě, je možné mít přehled o zadané gramatice napříč jednotlivými podstránkami. Uživateli se přitom jeví každá stránka jako jedna a ne jako 2 stránky poskládané. Pod tímto vstupním polem se pak nacházejí 4 možnosti, které lze s BG dělat. Pro představu

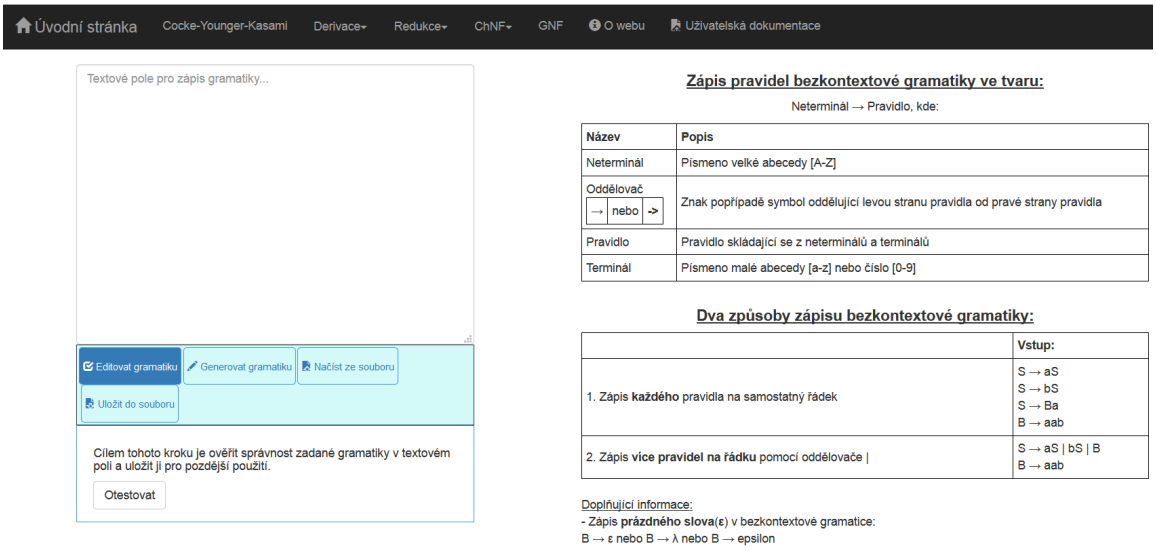
[4] `https://twitter.com/`

na obrázku 7 je zobrazena úvodní stránka, na které lze vidět navigační lištu a kde obsah levé strany odpovídá té popisované. Hlavní funkce této stránky jsou:

1. Uložení BG do zařízení (PC, mobil, ...).
2. Načtení BG ze zařízení zpět do stránek (dříve uložené BG).
3. Editování BG na stránce a kde tato možnost také slouží k otestování této BG na vstupu.
4. Generování BG podle počtu terminálů, neterminálů a pravidel (typu, délky pravých stran), které si uživatel nakliká.

7.3.1 Úvodní stránka

Úvodní stránka popisuje v krátkosti, co je to terminál, neterminál a jakým způsobem se zapisují pravidla v bezkontextové gramatice. Kromě toho je zde popsán způsob, jakým se zapisuje bezkontextová gramatika na výukové stránce, aby umožnila uživateli zadat i jeho vlastní vstup bez chyb a to ve správném tvaru. Dále je zde uvedena informace o tom, jak zapsat epsilon (ε) pravidla v BG. Na závěr v této kapitole (viz obrázek 7) je zobrazena úvodní stránka, která se zobrazí ve výchozím stavu při návštěvě stránek a tedy bez zadání žádné z jejich podstránek.

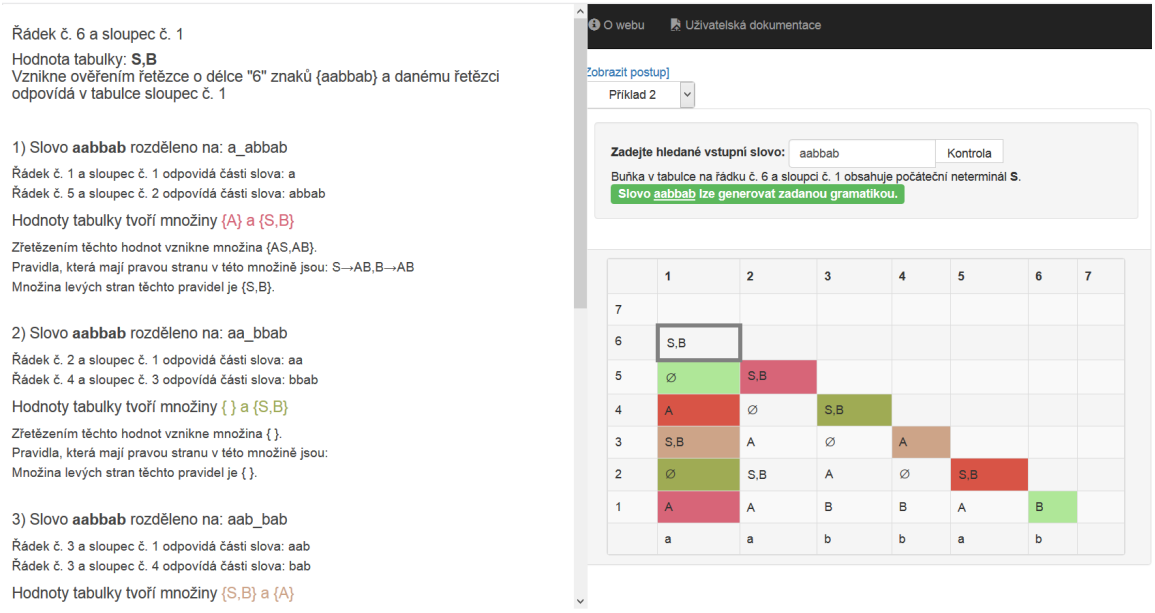


Obrázek 7: Úvodní stránka webové aplikace

7.3.2 Cocke-Younger Kasami

Cocke-Younger Kasami (CYK) stránka obsahuje 5 předpřipravených gramatik s kterými lze algoritmus CYK vyzkoušet bez nutnosti zadávat vlastní gramatiku na vstupu. Druhým způsobem je zadat vlastní BG v levé části stránky a vstupní slovo pro které se zde nachází textové pole

na stránce, které potvrdí pak tlačítkem „Kontrola“. Postup CYK algoritmu byl popsán v kapitole 4.5 a tento algoritmus provede a vrátí odpověď ANO/NE, jestli lze nebo nelze generovat zadanou gramatiku v ChNF zadané slovo. Součástí je CYK tabulka a vysouvací menu v levé části stránky obrazovky obsahující popis, jakým byla tabulka vyplněna. Dané vysouvací menu se zobrazuje v případě, že uživatel klikne na nějakou buňku v tabulce týkajícího se řešení příkladu. Obsahem vysouvacího menu je popis řešení, jakým byla daná buňka v tabulce vyplněna. Pokud by při algoritmu došlo k problémům, byl by na ně uživatel upozorněn formou chybové hlášky. Problém by mohl např. nastat, kdyby vstupní gramatika nebyla v ChNF nebo slovo neobsahovalo pouze terminální symboly. CYK stránky tedy tvoří vstupní pole pro vstupní řetězec (slovo), seznam předpřipravených příkladů a dále tlačítko, kterým se provádí kontrola. Obrázek 8 zobrazuje stránku s CYK algoritmem.



Obrázek 8: Stránka algoritmus CYK na webové aplikace

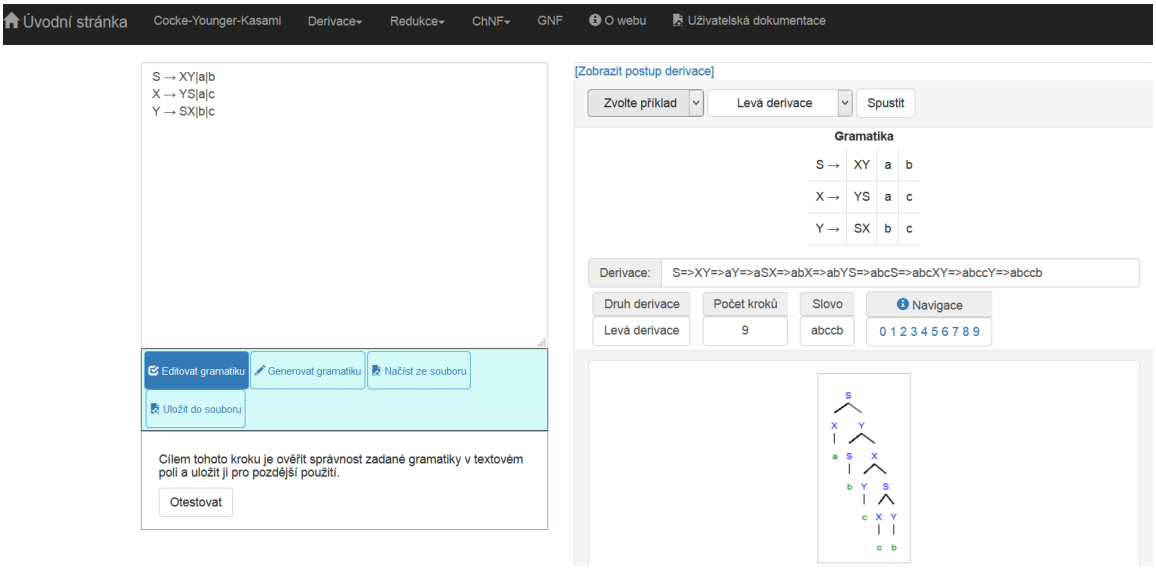
7.3.3 Derivace

Derivace na výukových stránkách se skládá ze 2 částí. První část je uživatelsky řízená derivace, kde uživatel sám vybírá, které pravidlo použije v derivaci (levá, pravá). Druhou částí je zobrazení derivace zadaného slova za pomocí Earleyho algoritmu a součástí řešení je i derivační strom.

7.3.4 Uživatelsky řízená derivace

Uživatelsky řízená derivace na stránce obsahuje 5 předpřipravených příkladů, které zobrazí už hotovou derivaci slova bez nutnosti zadávat vlastní vstup. V případě tohoto výběru se uživateli zobrazí již vyřešená derivace, proto je zde možnost vrátit se o několik kroků zpět v derivaci a zkusit derivovat jiné slovo nebo derivaci začít od začátku znova (zresetovat) pomocí „Navigace“,

ale nyní v derivaci. Výběr dalších pravidel probíhá formou klikání na pravidla BG na stránce, které se zobrazí při výběru příkladu nebo po odkliknutí tlačítka „Spustit“. Při zadávání vlastní gramatiky je nutné ještě vybrat typ derivace na stránce a pak teprve spustit derivaci. V případě, že by např. uživatel v průběhu derivace vybral pravidlo, které obsahuje na levé straně neterminál, který není další v pořadí viz. typ derivace, je uživatel na to upozorněn formou chybové hlášky. Součástí derivace je zobrazení konstrukce derivačního stromu za pomocí JavaScript doplňku **Syntax Tree Generator**[5] od autora Miles Shang. Stránka s uživatelsky řízenou derivaci je zobrazena na obrázku 9.

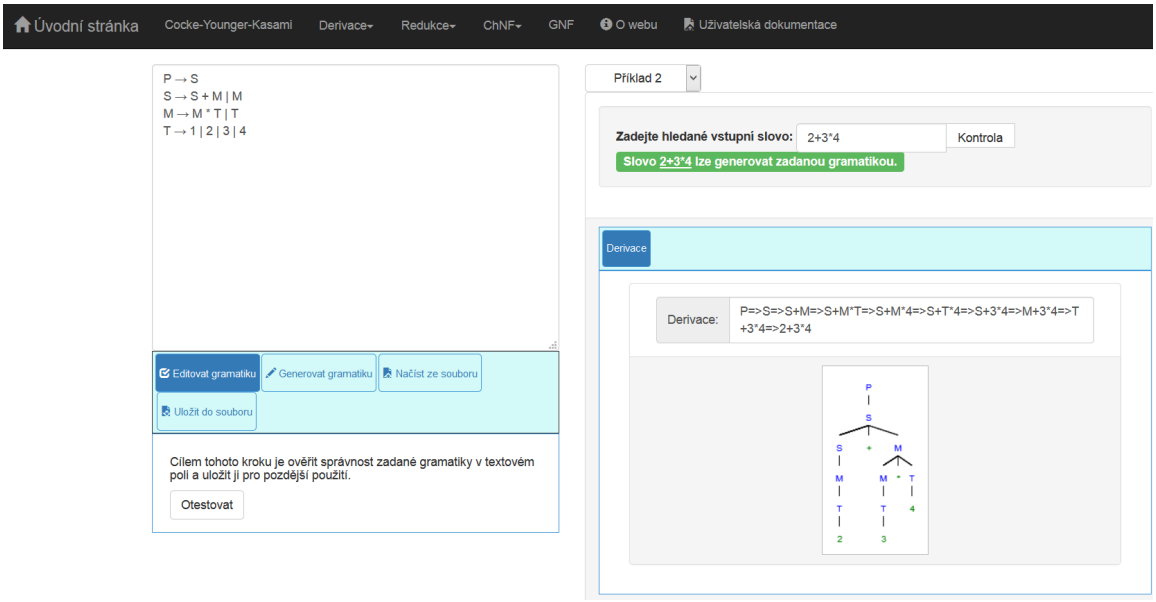


Obrázek 9: Stránka s uživatelsky řízenou derivaci na webové aplikace

7.3.5 Zobrazení derivace zadaného slova

Zobrazení derivace zadaného slova obsahuje až 16 předpřipravených příkladů bez nutnosti uživatele zadávat vlastní vstup. Zobrazení derivace je docíleno za pomocí Earleyho algoritmu popsaném v kapitole 4.6. Kromě zobrazení derivace je součástí řešení algoritmu na stránce také zobrazení derivačního stromu. Derivační strom je zobrazován díky **Syntax Tree Generator** zmíněném v kapitole 7.3.4. Samotné stránky obsahují vstupní pole pro ověřované (testované) slovo, které se pak potvrzuje tlačítkem „Kontrola“, aby se provedl daný algoritmus. Pokud uživatel chce vyzkoušet vlastní gramatiku, zadá do vstupního pole pro gramatiku vlastní vstup a potvrdí tlačítkem „Kontrola“. Postup derivace a derivační slovo je zobrazené jen v případě, že k takovému slovu lze dojít derivací. Stránka je zobrazena na obrázku 10.

[5] `https://github.com/mshang/syntree`

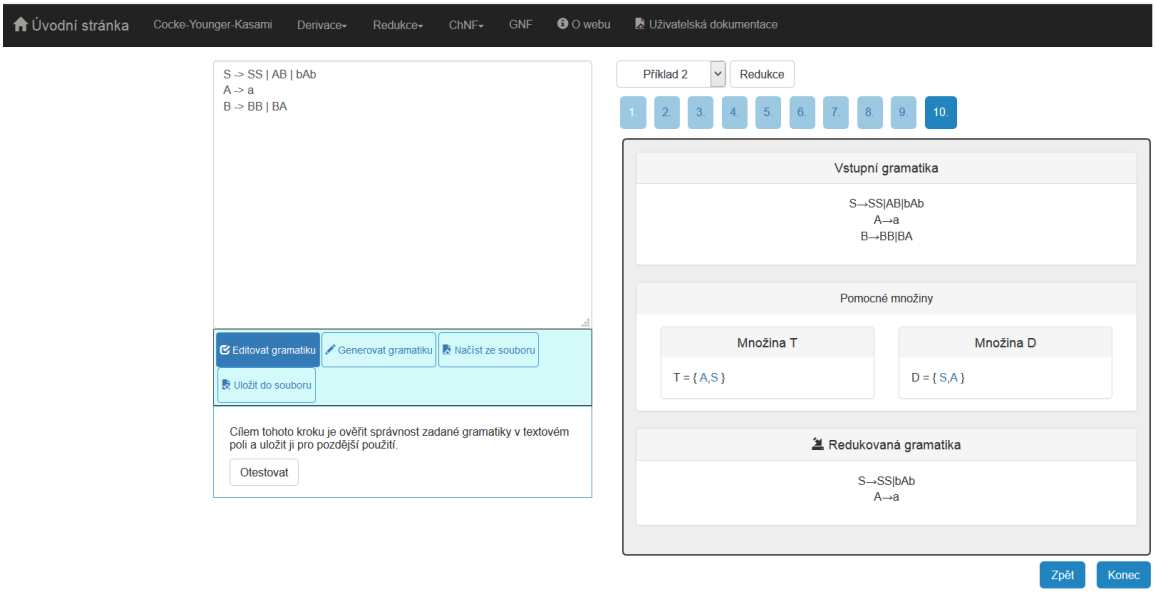


Obrázek 10: Stránka s Earleyho algoritmem na webové aplikace

7.3.6 Redukce

Redukce gramatiky obsahuje 5 předpřipravených příkladů bez nutnosti uživatele zadat vlastní gramatiku na vstupu (viz obrázek 11). Algoritmus na redukování BG byl již popsán v kapitole 4.1. Pro redukci gramatiky stačí vybrat jednu z předpřipravených gramatik a nebo zadat (vložit) vlastní a potvrdit tlačítkem „Redukce“. Výstupem na stránce je postup redukce BG za pomocí JavaScript doplňku jquery.steps[6].

[6] `https://github.com/rstaib/jquery-steps`



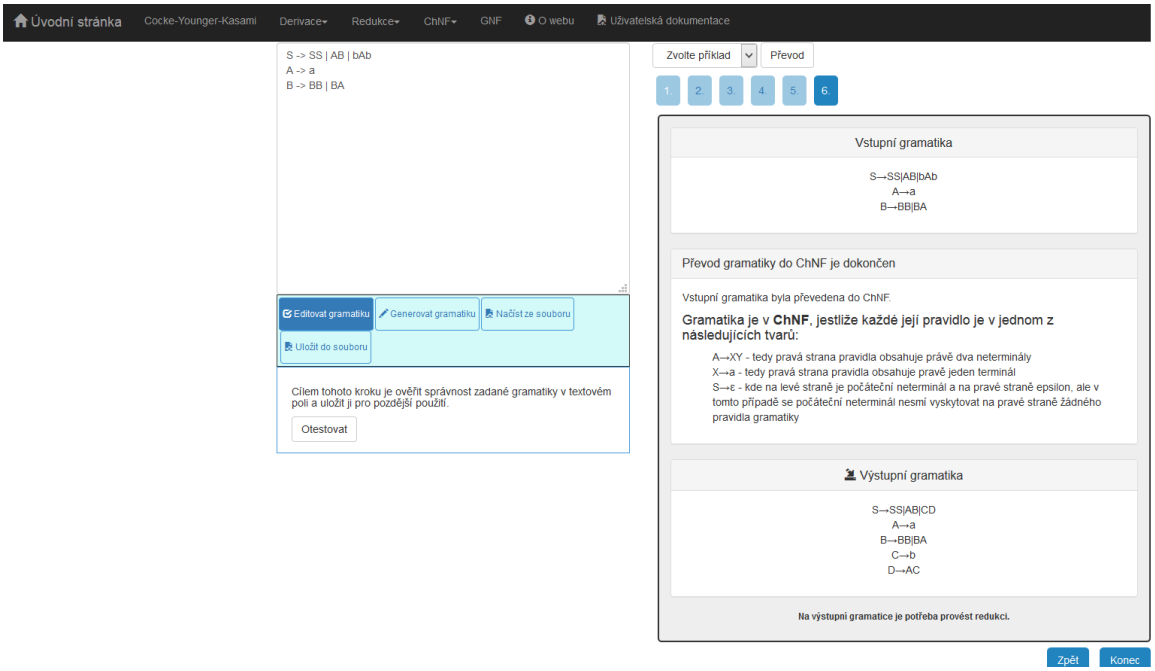
Obrázek 11: Stránka s redukci BG na webové aplikace

7.3.7 Testování redukce gramatiky

Testování redukce gramatiky spočívá v tom, že si uživatel vybere jeden z předpřipravených příkladů a pomocí výběru a zakliknutí správných checkboxů (zatrhávajících políček) podle popisku u nich pak dojde k jejich ověření správnosti po stisknutí tlačítka „Vyhodnocení“. Po vyhodnocení se zobrazí správné řešení. V případě chyb se zobrazí, které zatrhávající políčko bylo špatně vybrané.

7.3.8 ChNF

Převod do ChNF obsahuje 5 předpřipravených příkladů bez nutnosti uživatele zadat vlastní gramatiku na vstupu (viz obrázek 12). Jak algoritmus BG převádí do ChNF již bylo uvedeno v kapitole 4.3. Pokud uživatel bude potřebovat zadat vlastní vstup, tak ho může zadat, ale pak musí potvrdit převod do ChNF tlačítkem „Převod“. Postup převodu je zobrazen pomocí stejného JavaScript doplňku jako v případě redukce a to pomocí **jquery.steps**.



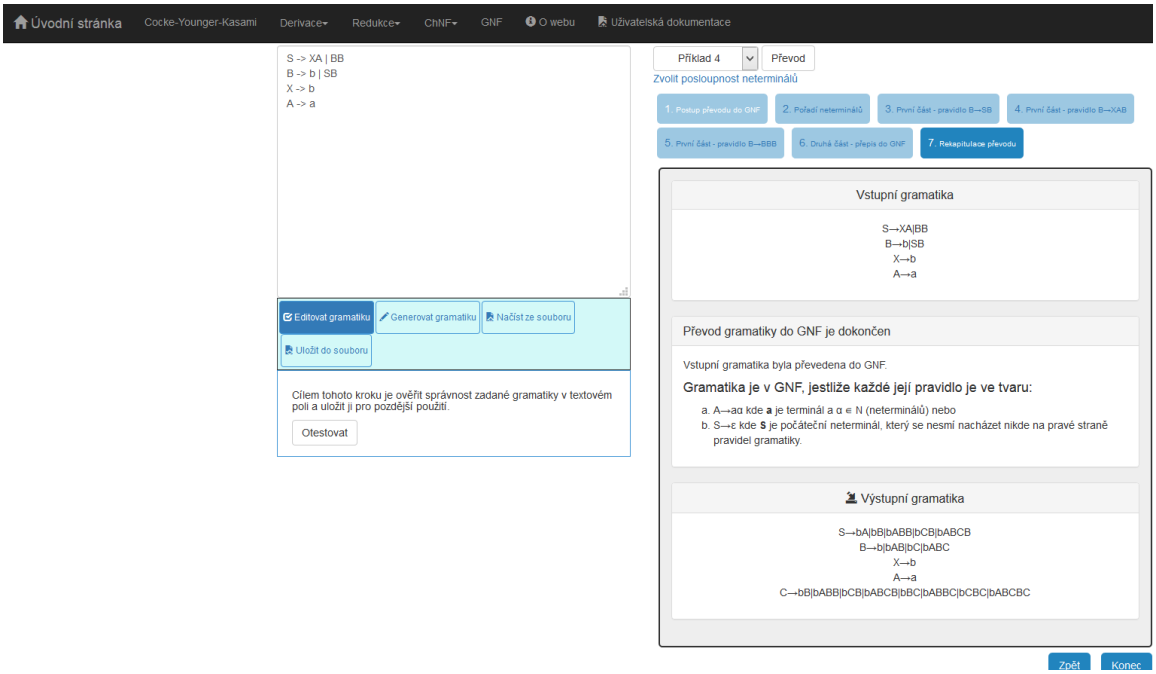
Obrázek 12: Stránka s převodem BG do ChNF

7.3.9 Testování odstraňování epsilon a unit pravidel

Testování odstraňování epsilon a unit pravidel slouží k ověření a vyzkoušení znalostí a zda-li uživatel, který to zkouší, tomuto „procesu“ odstraňování (nahrazování pravidel) rozumí. Uživatel si zvolí jeden z předpřipravených příkladů a následně vyřeší daný příklad. Po vyhodnocení se zobrazí správně řešení a dobře/špatně vybrané odpovědi.

7.3.10 GNF

Převod do GNF obsahuje 5 předpřipravených příkladů bez nutnosti uživatele zadat vlastní gramatiku na vstupu (viz obrázek 12). Jak algoritmus převádí gramatiku do GNF je v kapitole 4.4. Pokud uživatel bude chtít zadat vlastní vstup, tak převod samotný musí začít potvrzením tlačítka „Převod“. Postup převodu je zobrazen pomocí stejného JavaScript doplňku jako v případě již zmíněné redukce a převodu do ChNF a to pomocí **jquery.steps**.



Obrázek 13: Stránka s převodem do GNF

7.3.11 O webu

O webu obsahuje výčet všech algoritmů a funkcí na výukové stránce s možnosti rychlé navigace po nich. Stránky díky těmto odkazům v textu umožňují další rychlý přesun po algoritmech a funkcích na stránce.

7.3.12 Uživatelská dokumentace

Uživatelská dokumentace je dokument pdf uložený na straně serveru a seznamuje uživatele s výukovými stránkami.

- Popisuje požadavky na uživatele přesněji řečeno na internetový prohlížeč, který používá při návštěvě výukových stránek.
  - Je nutné mít povolené Cookies a JavasSript pro správnou funkčnost a zobrazování stránek.
- Navigaci na stránce a ukázkový zápis gramatiky a její omezení při zápisu.
- Ukázka použití každého algoritmu (podstránky) na výukové stránce.

7.4 Systémové požadavky

Systémové požadavky na webovou aplikaci pro sestavení a testování jsou následující.

- Windows 7, 8.1, 10 (x64)
- Visual Studio 2013 a výše
- IIS Express
- .NET Framework 4.5 a výše
- CPU dostatečně výkonné a dostatečnou velikost paměti RAM (4GB+)

Webová aplikace byla vyvíjená a testována na sestavě uvedené v seznamu níže.

- Windows 8.1 (x64)
- Intel(R) Core(TM) i7-4720HQ CPU 2.60Ghz
- NVIDIA GeForce GTX 960M
- 8GB RAM

7.5 Sestavení webové aplikace

Na přiloženém CD a nebo na stejném místě v jakém se nachází dokument, je archív s názvem „zdrojové_kódy.zip“. V daném archívu jsou uloženy veškeré zdrojové kódy z kterých se webová aplikace skládá. K sestavení neboli zkompilování zdrojových kódů, je doporučené tento archív nejdříve přesunout na disk a až pak rozbalit. Po rozbalení archívu je potřeba najít soubor „Context-Free Grammar.csproj“, který se nachází ve složce:
\zdrojové_kódy\Web\ContextFreeGrammar. Tento soubor je potřeba otevřít za pomocí Visual Studia, ale než se tak provede, je nutné splnit následující podmínky:

1. Mít nainstalovaný .NET Framework 4.5 a nebo výše.
2. Mít nainstalované Visual Studio 2013 (doporučené) a nebo výše.
3. Mít funkční připojení k internetu z důvodu dodatečného stáhnutí chybějících balíčků (jQuery, bootstrap, atd.), které proběhne automaticky při prvním otevření projektu v Visual Studiu.
4. Mít nainstalovaný Internet Information Services zkáceně IIS Express (pro testování a běh webové aplikace).

Po splnění těchto podmínek zbývá otevřít zmiňovaný soubor „Context-Free Grammar.csproj“ v Visual Studiu. Ve Visual Studiu zvolit operaci „Spustit“ (Build) pro hlavní projekt (ve výchozím nastavení Context-Free Grammar). Jakmile je projekt sestavený, je na výběr mezi spuštěním webové aplikace přímo nebo jí „Publikovat“. První možnost spočívá ve stisknutí tlačítka „Spustit“ (Run) kdy se otevře prohlížeč s danou webovou stránkou. Druhá možnost se provede najetím myší na projekt „CFG_WebApp“ a otevřením menu pravým tlačítkem myši a následným zvolením možnosti „Publikovat“. Provedenou akcí se otevře nové dialogové okno s přihlášením na vzdálený server tam, kde stránka je nebo teprve bude publikována, a nebo výstup lze např. uložit do složky v PC. Pokud zvolíme nahrání na server, je dobré si předtím zkontrolovat, že vše funguje bez problémů, aby se nestalo, že např. stránky pak nepůjdou načíst (přestanou fungovat). V případě sestavování webové aplikace a následného přesouvání na vzdálený server, je v některých případech dodatečně ještě v konfiguračním souboru aplikace nutné přidat plná oprávnění. Jak toho docílit je v kapitole 7.6.

7.6 Nasazení webové aplikace

Před nasazením webové aplikace na server, je potřeba zjistit, jestli tento hostující server podporuje ASP.NET stránky. Druhou možností je nasadit webovou aplikaci na PC (lokálně) a to za pomocí IIS Express a .NET Frameworku 4.5 a výše. V případě vybrání lokálního nasazení je nutné buď zkompilovat (sestavit) webovou aplikaci a nechat ji tzv. „publikovat“. Tento proces byl již popsán v kapitole 7.5. Druhou možností je použít již zkompilovaný kód v archívu, který je už připravený k nasazení na server nebo v PC (lokálně). Název archívu je „WebApp.zip“ a nachází se v přiloženém CD u diplomové práce nebo na stejném místě, jako byl nahrán dokument. Při vybrání archívu je potřeba ho nejdříve rozbalit a až pak nahrát s jeho obsahem na server nebo ho spustit pomocí IIS Express v PC. Jak aplikaci spustit nebo nahrát na server je popsáno v seznamu níže pod tímto textem.

(a) Nasazení (spuštění) webových stránek v PC s operačním systémem Windows (testováno na 7, 8.1)

   (1) Zkopírovat soubor „webové-stránky.zip“ do zařízení (PC) a rozbalit ho.

   (2) Nainstalovat IIS Express[7].

   (3) Spustit příkazový řádek (cmd) a vložit do něj následující příkaz:

   ```
   start iisexpress /path:"PATH" /port:9090 /clr:v4.0 /systray:true
   ```

   , kde „PATH“ je potřeba přepsat na složku umístění rozbaleného adresáře (např. "C:\").

   (4) Potvrdit příkaz a počkat na spuštění IIS Express.

[7] `https://www.microsoft.com/cs-cz/download/details.aspx?id=48264`

(5) Otevřít webový prohlížeč a zadat do vyhledávacího řádku adresu: 127.0.0.1:9090, kde 9090 je port, který se musí shodovat s tím, který byl zadán v příkazu předtím ke spuštění IIS Expressu.

(b) Nasazení webových stránek na server

(1) Zkontrolovat, že daný server podporuje ASP.NET stránky.

(2) Nahrát nejčastěji obsah rozbaleného archívu do složky na serveru.

(3) Pokud vše proběhlo bez problémů, tak stačí otevřít prohlížeč a zadat adresu stránky.

(c) Použít již zprovozněné (běžící) stránky na serveru přes odkaz: `http://cfgrammar.aspone.cz/` (Upozornění: tyto stránky jsou hostovány zdarma a může se tedy stát, že v budoucnu např. při čtení dokumentu se stránky již zde nemusí nacházet).

8 Předpřipravené příklady

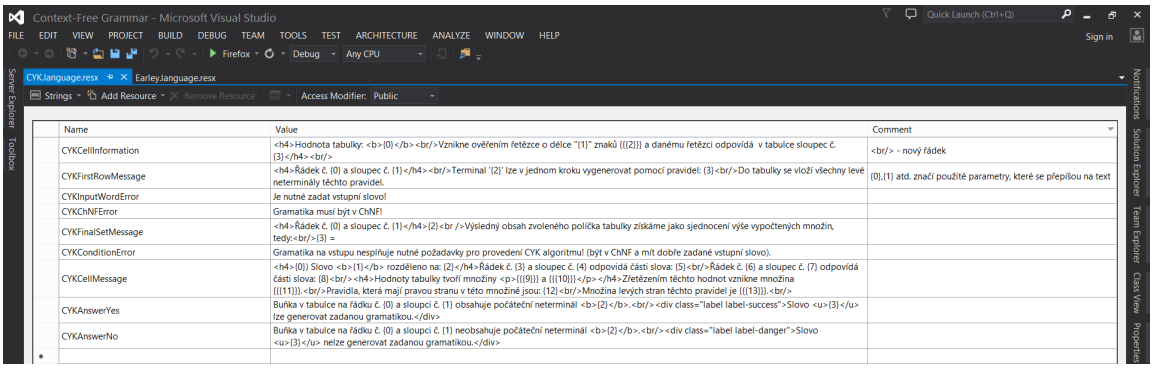
Předpřipravené příklady bezkontextové gramatiky se nachází ve složce:
„\Context-Free Grammar\Context-Free Grammar\Examples\“. Soubory nabízejí možnost změnit, odstranit nebo přidat nové příklady na výukové stránky u konkrétních podstránek s algoritmy. Důležitá informace je ta, že každý příklad je potřeba napsat na jeden řádek a v případě potřeby zadat ověřované (hledané) slovo nebo jinou informaci, tak je potřeba je od sebe oddělit čárkou. Pokud by tedy chtěl správce stránek přidat nový příklad např. k algoritmu CYK otevře soubor „CYKPageExamples“ v textovém editoru. V souboru začíná první řádek vždy znaky \\ a značí začátek komentáře. Komentář popisuje, jak se mají příklady do souboru zadávat. Dalším pomocným indikátorem jsou již připravené předpřipravené příklady. V další části si ukážeme přidání nového příkladu na nový (prázdný) řádek v souboru, který vypadá, takto:

- $S \rightarrow SC|SA|SB\backslash nC \rightarrow AC|c\backslash nA \rightarrow a|AA\backslash nB \rightarrow BB|b$, cacca

Na daném řádku je zapsána BG, která je oddělena čárkou (,) od vstupního slova **cacca**. Tomuto zápisu odpovídá BG ve tvaru:
$S->SC|CA|SB$
$C->CS|c$
$A->a|AA$
$B->BB|b$
Jakmile soubor nebo soubory uložíme, tak předpřipravené příklady u jednotlivých algoritmů se při dalším obnovení stránky aktualizují. To znamená, že v dalším načtení bude zobrazovat i ty příklady, které byly přidány nově. To platí i v případě, že příklady na řádku smažeme nebo upravíme. V případě úprav souborů doporučuji upravovat je mimo server (originálního umístění souboru) z důvodu možného způsobení uzamknutí (znepřístupnění) souborů. Znepřístupnění souborů vede k nezobrazení předpřipravených příkladů u daného algoritmu na podstránce. Je tedy možné bezstarostně používat ostatní algoritmy s předpřipravenými příklady, ale není to ideální řešení.

9 Texty na stránkách a v postupu řešení algoritmů

Texty na stránkách a v postupu řešení algoritmů jsou ve webové aplikaci řešeny pomocí „Resources.resx“ souborů. Konkrétně ve složce s CFG_ServiceLayer projektem se nachází složka *Properties* a v ní tyto soubory (.resx). K úpravě těchto souborů lze použít jakýkoliv textový editor, ale stačí i Visual Studio. Výhodou tohoto ukládání řetězců do Resource souborů je ten, že programátor nebo jiná osoba pak nemusí hledat na více místech ve zdrojových kódech (jednotlivých třídách), kde se tyto řetězce nacházejí. Místo toho přistupuje do souborů (.resx) a upravuje je na tomto místě. Každý takový soubor je použit pro jeden implementovaný algoritmus starající se o zobrazování textů. Řetězce v souborech jsou rozlišeny podle tzv. názvu (klíče, identifikátoru). Název slouží k identifikaci řetězce a k vložení do zdrojového kódu. Při spuštění a zavolání konkretního řádku se vrací zadaný obsah řetězce (value), který mu byl přiřazen. Další výhodou je, že popisy algoritmů můžeme snadno přeložit do jiných jazyků nebo je upravovat. Jak takový soubor vypadá při otevření ve Visual Studiu je ukázané na obrázku 14. Soubor, který je zde zobrazen, slouží konkrétně pro algoritmus CYK. Na obrázku si lze všimnout, že, kromě názvu a hodnoty řetězce, je zde možnost i zadat komentář pro každý řádek.



Obrázek 14: Řešení textů u algoritmů

10 Závěr

10.1 Zhodnocení

Cílem bylo vytvořit komponentu, která by byla spolu s dalšíma nasazena na server týkající se teoretické informatiky. Tato konkretní komponenta by sloužila studentům k naučení základů bezkontextových gramatik. Výsledkem jsou výukové stránky napsané v ASP.NET Web Forms. Chtěl bych v krátkosti zmínit, že i když jsem se v průběhu tvoření webové aplikace potýkal s problémy a několikrát se musel vydat jinou cestou, ať už z důvodu špatné volby technologie, tak i omezení ze strany serveru, nebo špatné prvotní úvahy algoritmu při implementaci, se mi nakonec podařilo splnit zadané požadavky a dokonce i některé přidat navíc. Konkrétně jsem přidal možnost generovat BG a redukovat BG, ale formou testování pomocí zaklikávání správných odpovědí a to zaškrtáváním okýnek (checkboxů) u předpřipravených příkladů. Kromě těchto jsem přidal i možnost testování odstraňování epsilon a unit pravidel, které je jinak součástí převodu do ChNF. V menší podkapitole v závěru (viz podkapitola 10.2) jsou ještě krátce uvedeny nápady na možná vylepšení výukových stránek do budoucna.

10.2 Rozšíření výukových stránek

Rozšíření výukového serveru by bylo možné, ať už jen jeho části nebo celku. Za úvahu by stálo rozšířit aplikaci o tyto možnosti nebo upravit některá dosavadní řešení.

1. Přidat přihlášení uživatele (administrátora) do stránek s možnosti editovat a ukládat předpřipravené příklady přes databázi.
2. Upravit uživatelsky řízené derivace tak, aby nepotřebovala v průběhu derivace se dotazovat na další krok serveru a tedy přepsat derivaci nejlépe do JS.
3. Rozšířit aplikaci o další algoritmy týkající se BG.
4. Upravit, aby nebylo potřeba při změně textů u implementovaných algoritmů znovu sestavovat celou aplikaci, ale jen nahrát upravené texty.

Literatura

[1] Formální jazyky. *Matematika pro střední a základní školy* [online]. [cit. 2017-04-16]. Dostupné z: `https://matematika.cz/formalni-jazyky`

[2] HORDĚJČUK, Vojta. Formální gramatika. *Matematika* [online]. [cit. 2017-04-18]. Dostupné z: `http://voho.eu/wiki/formalni-gramatika/`

[3] Chomsky hierarchy. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2017-04-18]. Dostupné z: `https://en.wikipedia.org/wiki/Chomsky_hierarchy`

[4] JANČAR, Petr. *Teoretická informatika*, [online]. Ostrava: Vysoká škola báňská - Technická univerzita, 2008 [cit. 2017-04-16]. ISBN 978-80-248-1487-2. Dostupné z: `http://www.cs.vsb.cz/jancar/TEORET-INF/ti-text.2010-01-20.pdf`

[5] Redukované gramatiky. *Animace k předmětům teoretické informatiky: Bezkontextové gramatiky a zásobníkové automaty* [online]. [cit. 2017-04-18]. Dostupné z: `http://www.cs.vsb.cz/kot/soubory_animaci/a-bg_redukce.pdf`

[6] Chomsky normal form. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2017-04-18]. Dostupné z: `https://en.wikipedia.org/wiki/Chomsky_normal_form`

[7] Greibach Normal Form. *Formal Languages and Automata Theory: Properties of Context-free Languages* [online]. 2011 [cit. 2017-04-18]. Dostupné z: `http://www.iitg.ernet.in/gkd/ma513/oct/oct18/note.pdf`

[8] CYK algorithm. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2017-04-18]. Dostupné z: `https://en.wikipedia.org/wiki/CYK_algorithm`

[9] LANGE, Martin a Hans LEIß. *To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm* [online]. 2009 [cit. 2017-11-28]. Dostupné z: `https://www.informaticadidactica.de/index.php?page=LangeLeiss2009`

[10] Earley parser. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2017-11-28]. Dostupné z: `https://en.wikipedia.org/wiki/Earley_parser`

[11] AYCOCK, John a R., Nigel HORSPOOL. *Practical Earley Parsing* [online]. 2001 [cit. 2017-11-28]. Dostupné z: `http://webhome.cs.uvic.ca/~nigelh/Publications/PracticalEarleyParsing.pdf`

[12] ASP.NET Web Forms | The ASP.NET Site. *ASP.NET / The ASP.NET Site* [online]. [cit. 2018-04-22]. Dostupné z:
`https://www.asp.net/web-forms`

[13] OTTO, Mark a Jacob THORNTON. Bootstrap. *Bootstrap · The most popular HTML, CSS, and JS library in the world.* [online]. [cit. 2018-02-17]. Dostupné z:
`https://getbootstrap.com/`

[14] JavaScript. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2018-02-17]. Dostupné z:
`https://en.wikipedia.org/wiki/JavaScript`

[15] JQuery. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2018-04-22]. Dostupné z:
`https://en.wikipedia.org/wiki/JQuery`

[16] ASP.NET Cookies Overview. *Learn to Develop with Microsoft Developer Network | MSDN* [online]. [cit. 2018-04-22]. Dostupné z:
`https://msdn.microsoft.com/en-us/library/ms178194.aspx`

[17] SHANG, Miles. Syntree. *GitHub* [online]. [cit. 2018-04-22]. Dostupné z:
`https://github.com/mshang/syntree`

[18] SCHWARTZ, Adam. Vex. *GitHub* [online]. [cit. 2018-04-22]. Dostupné z:
`https://github.com/hubspot/vex`

[19] J. STAIB, Rafael. JQuery Steps. *GitHub* [online]. [cit. 2018-04-22]. Dostupné z:
`https://github.com/rstaib/jquery-steps`

A Struktura přiloženého CD

Struktura přiloženého CD je následovná.

1. \www\WebApp.zip
    - Archív obsahující připravené stránky k nasazení na server.
2. \www\zdrojové kódy.zip
    - Archív obsahující zdrojové kódy webové aplikace (BG, algoritmy, servisní vrstvu).
3. \Readme.txt
    - Soubor se slovním popisem struktury přiloženého CD.
4. \Uživatelská dokumentace.pdf.
    - Soubor .pdf s uživatelskou dokumentací výukových stránek.